# SIEMENS

SIMATIC NET

# Programmierschnittstelle DP-Base für CP 5613/CP 5614

**Handbuch**




**C79000-G8900-C139-04**

**Ausgabe 05/2000**

## Sicherheitstechnische Hinweise

Dieses Handbuch enthält Hinweise, die Sie zu Ihrer persönlichen Sicherheit sowie zur Vermeidung von Sachschäden beachten müssen. Die Hinweise sind durch ein Warndreieck hervorgehoben und je nach Gefährdungsgrad folgendermaßen dargestellt:



**Gefahr**

bedeutet, dass Tod, schwere Körperverletzung oder erheblicher Sachschaden eintreten **werden**, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.



**Warnung**

bedeutet, dass Tod, schwere Körperverletzung oder erheblicher Sachschaden eintreten **können**, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.



**Vorsicht**

bedeutet, dass eine leichte Körperverletzung oder ein Sachschaden eintreten können, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

**Hinweis**

ist eine wichtige Information über das Produkt, die Handhabung des Produktes oder den jeweiligen Teil der Dokumentation, auf den besonders aufmerksam gemacht werden soll.

## Qualifiziertes Personal

Inbetriebsetzung und Betrieb eines Gerätes dürfen nur von qualifiziertem Personal vorgenommen werden. Qualifiziertes Personal im Sinne der sicherheitstechnischen Hinweise dieses Handbuchs sind Personen, die die Berechtigung haben, Geräte, Systeme und Stromkreise gemäß den Standards der Sicherheitstechnik in Betrieb zu nehmen, zu erden und zu kennzeichnen.

## Bestimmungsgemäßer Gebrauch

Bitte beachten Sie folgendes:



**Warnung**

Das Gerät darf nur für die im Katalog und in der technischen Beschreibung vorgesehenen Einsatzfälle und nur in Verbindung mit von Siemens empfohlenen bzw. zugelassenen Fremdgeräten und -komponenten verwendet werden.

Der einwandfreie und sichere Betrieb des Produktes setzt sachgemäßen Transport, sachgemäße Lagerung, Aufstellung und Montage sowie sorgfältige Bedienung und Instandhaltung voraus.

## Marken

SIMATIC® und SIMATIC NET® sind Marken der Siemens AG.

Die übrigen Bezeichnungen in dieser Schrift können Marken sein, deren Benutzung durch Dritte für deren Zwecke die Rechte der Inhaber verletzen können.



**Haftungsausschluß**

Wir haben den Inhalt der Druckschrift auf Übereinstimmung mit der beschriebenen Hard- und Software geprüft. Dennoch können Abweichungen nicht ausgeschlossen werden, so daß wir für die vollständige Übereinstimmung keine Gewähr übernehmen. Die Angaben in dieser Druckschrift werden regelmäßig überprüft und notwendige Korrekturen sind in den nachfolgenden Auflagen enthalten. Für Verbesserungsvorschläge sind wir dankbar.

Siemens AG
Bereich Automatisierungs- und Antriebstechnik
Postfach 48 48, D-90327 Nürnberg

C79000-G8900-C139-04


Siemens Aktiengesellschaft

Printed in the Federal Republic of Germany

# Vorwort

## Zweck des Handbuch

Dieses Handbuch unterstützt Sie bei der Erstellung von Anwenderprogrammen für die DP-Programmierschnittstelle des CP 5613/CP 5614.

Wir setzen voraus, dass Sie mit der Erstellung von Anwenderprogrammen in der Programmiersprache „C“ unter Windows NT vertraut sind.

## Gültigkeitsbereich dieses Handbuchs

Dieses Handbuch ist für folgende Software-Versionen gültig:

- CP 5613/CP 5614 (DP-Base V1.0)
- CP 5613/CP 5614 (DP-Base V1.1)
- CP 5613/CP 5614 (DP-Base V1.2)

## Wegweiser

Um Ihnen den schnellen Zugriff auf spezielle Informationen zu erleichtern, enthält das Handbuch folgende Zugriffshilfen:

- Am Anfang des Handbuchs finden Sie ein vollständiges Gesamtinhaltsverzeichnis.
- Am Ende des Handbuchs finden Sie ein ausführliches Stichwortverzeichnis, welches Ihnen den schnellen Zugriff auf die gewünschte Information ermöglicht.
- Am Schluss finden Sie ein Glossar, in welchem wichtige Fachbegriffe definiert sind, die im Handbuch verwendet werden.

# Inhaltsverzeichnis

# Übersicht Erstellen einer DP-Anwendung

# 1

In diesem Kapitel empfehlen wir Ihnen ein schrittweises Vorgehen zum Erstellen eines DP-Anwenderprogramms auf Basis der DP-Programmierschnittstelle des CP 5613 und CP 5614, genannt „DP-Base“. Die Schritte beginnen mit dem Vermitteln der Grundlagen von PROFIBUS-DP und enden mit dem Testen Ihrer Anwendung.

Die DP-Base-Programmierschnittstelle erlaubt Direktzugriffe auf das DP-Prozessabbild im Dualport RAM der Baugruppe. Die DP-Base-Programmierschnittstelle ist deshalb nicht mit der DP-Programmierschnittstelle des CP 5412 (A2), des CP 5511 und des CP 5611 kompatibel.

## Vorgehensweise

So kommen Sie am einfachsten und schnellsten zum Ziel:

| Schritt | Beschreibung |
|---|---|
| 1 | Lernen Sie die Grundprinzipen von PROFIBUS DP kennen. Lesen Sie dazu das nachfolgende Kapitel 2 „Übersicht PROFIBUS DP“. |
| 2 | Lernen Sie die wesentlichen Eigenschaften der DP-Base-Programmierschnittstelle des CP 5613 und CP 5614 kennen. Lesen Sie dazu das nachfolgende Kapitel 3 „Übersicht “. |
| 3 | Machen Sie sich mit dem Inhalt des Unterverzeichnisses „prog“ in Ihrem Installationsverzeichnis vertraut, so dass Sie für die nachfolgenden Schritte wissen, welche Teile vorhanden sind und wozu sie dienen.<br>Im einzelnen sind das:<br>• Die Datei „liesmich.txt“ mit neuesten, zusätzlichen Informationen und letzen Änderungen.<br>• Die C-Header-Datei „dp_5613.h“ mit den Funktionen und Datenstrukturen der DP-Base-Schnittstelle sowie „5613_ret.h“ mit den Return-Codes.<br>• Die Import-Libraries „dp_base.lib“ und „dps_base.lib“ zum Zubinden an Ihr Anwenderprogramm.<br>• Die Beispielprogramme und zugehörige Datenbasen im Unterverzeichnis „examples“. |
| 4 | Arbeiten Sie nun den Quelltext des Beispielprogramms „ExamEasy“ durch und schlagen Sie die Funktionen und Datenzugriffe im nachfolgenden Kapitel 4 „Beschreibung der einzelnen DP-Funktionen und Daten sowie Fehlercodes“ nach. |
| 5 | Ändern Sie das Beispielprogramm „ExamEasy“ entsprechend Ihrer Anlagenkonfiguration ab, indem Sie z. B. zusätzliche oder andere Slaves verwenden. Übersetzen und binden Sie das Beispielprogramm und probieren Sie es aus. Dazu müssen Sie eventuell die „ExamEasy“-Beispieldatenbasis erweitern (mit dem COM PROFIBUS). |
| 6 | Arbeiten Sie nun den Quelltext des Beispielprogramms „ExamComp“ durch, modifizieren Sie das Programm und erproben Sie es mit einer erweiterten Beispieldatenbasis. |
| 7 | Falls Sie die Slave-Funktionen des CP 5614 benutzen möchten, arbeiten Sie außerdem bitte das Beispiel „transfer“ durch, modifizieren Sie es entsprechend Ihren Anforderungen und erproben Sie es mit einer erweiterten Beispieldatenbasis. |
| 8 | Lesen Sie bitte die FAQ-Liste im Kapitel 5 durch, besonders die Checkliste für Programmierer und die Informationen zur Strukturierung Ihres Anwenderprogramms. |

Fortsetzung der Tabelle auf der nächsten Seite

Fortsetzung der Tabelle von der letzten Seite

| Schritt | Beschreibung |
|---|---|
| 9 | Erstellen Sie nun Ihr eigenes DP-Anwenderprogramm. Dazu können Sie das „ExamComp“-Beispiel als Grundlage verwenden. |
| 10 | Testen Sie Ihre Anwendung. Beachten Sie dazu bitte die Hilfestellungen und Tipps an der entsprechenden Stelle in der FAQ-Liste in Kapitel 5 sowie die in der Installationsanleitung beschriebenen Diagnosemöglichkeiten. |

# Übersicht PROFIBUS DP

# 2

In diesem Kapitel lernen Sie die Grundprinzipien von PROFIBUS DP kennen.

## 2.1 Positionierung von PROFIBUS DP

### PROFIBUS - Die weltweite Feldbusstrategie

Im Zuge von Kostenreduzierungen im Automatisierungsbereich werden speicherprogrammierbare Steuerungen (SPS), PCs, Antriebe, Messumformer und Sensoren immer stärker vernetzt. Weiter findet eine zunehmende Dezentralisierung von Steuerungsfunktionen durch Feldgeräte statt, die zum Informationsaustausch als gemeinsames Kommunikationsmedium Feldbusse nutzen.

Der Ruf nach einem offenen, herstellerunabhängigen Feldbussystem zur Investitionssicherung für die Anwender kann heute durch den PROFIBUS beantwortet werden. PROFIBUS ist ein Bussystem für die Kommunikation zwischen SPS oder PC und Feldgeräten, verankert in der europäischen Norm EN 50 170, Volume 2. Dies garantiert sowohl den Anwendern als auch Herstellern Investitionssicherheit und Offenheit für alle normgerechten Lösungen weltweit.

Als verschiedene Referenzen kann PROFIBUS mehr als 2 Millionen vernetzte Knoten in über 200.000 Anwenderprogrammen aufweisen. PROFIBUS ist damit der erfolgreichste offene Feldbus, der sich in vielen Anwendungen im Bereich der Fertigungsautomatisierung, Prozessautomatisierung, Antriebstechnik und Gebäudeautomatisierung bewährt hat.

Als übergreifendes Informationsforum für PROFIBUS-Hersteller und -Anwender agiert in diesem Zusammenhang die **P**ROFIBUS **N**utzer**o**rganisation (PNO), ein Zusammenschluss von mehr als 800 Anwendern, Herstellern und Beratern aus über 20 Ländern weltweit. Daraus resultierend sind heute für die PROFIBUS-Technik mehr als 1616 Produkte verfügbar.

Siemens unterstützt PROFIBUS als optimierte Feldbuslösung und sichere Investition für Anwender seit langer Zeit und bietet hierbei sowohl Produkte als auch komplette Systemlösungen an. Neben den Automatisierungssystemen (SPS) ergänzen Produkte wie Netzkomponenten, PC-Baugruppen und Feldgeräte für PROFIBUS das vielfältige Produktspektrum.

### Die Rolle des PC am PROFIBUS

Neben dem Trend der Dezentralisierung nimmt der Standard-PC als Automatisierungsgerät durch seine Performance-Steigerung und Verbreitung immer mehr an Bedeutung zu, besonders bei Steuerungsaufgaben und bei der Visualisierung.

## Die Vorteile von DP

PROFIBUS-DP ist für den schnellen Datenaustausch im Feldbusbereich konzipiert.

Dezentrale Peripheriegeräte sammeln die Eingabesignale vor Ort und übertragen sie über den Feldbus an die zentrale Steuerung im PG/PC. Umgekehrt sendet die zentrale Steuerung die Ausgabedaten an die dezentralen Peripheriegeräte.

Der Einsatz von PROFIBUS DP bewirkt eine erhebliche Reduzierung des Verkabelungsaufwands im Vergleich zur bisherigen direkten Verdrahtung der Komponenten.

## 2.2 Das Master-Slave-Konzept von PROFIBUS DP

### Dezentrale Peripherie

Die Dezentrale Peripherie (im folgenden mit DP abgekürzt) ermöglicht es, eine Vielzahl von analogen und digitalen Ein-/Ausgabebaugruppen dezentral und prozessnah einzusetzen.

### Busteilnehmerklassen

PROFIBUS-DP definiert zwei Klassen von Busteilnehmern: Slaves als Peripheriegeräte sind passive Busteilnehmer. Master (Klasse 1) sind aktive Busteilnehmer und steuern die Slaves.

### DP-Master Klasse 1

Mithilfe des CP 5613 oder CP 5614 kann der PC die Rolle des DP-Masters einnehmen. Je nach Anwendung kann diese Aufgabe z. B. auch von einer speicherprogrammierbaren Steuerung SIMATIC S7 übernommen werden. Nachfolgend wird der DP-Master Klasse 1 als DP-Master bezeichnet.

### DP-Master Klasse 2

Bei der Inbetriebnahme zur Konfiguration des DP-Systems oder zur Anlagebedienung im laufenden Betrieb können Master-Klasse-2-Geräte eingesetzt werden.

## DP-Slave

Ein DP-Slave ist ein Peripheriegerät, bei dem der Master Eingabeinformationen einliest und Ausgabeinformationen abgibt. Es sind auch Geräte möglich, die nur Eingabe- oder nur Ausgabeinformationen bereitstellen.

Slaves sind in der Regel sehr preiswert, weil die passive Teilnahme am Bus einfach realisierbar ist.

Abbildung 1 zeigt den prinzipiellen Aufbau und die Komponenten eines PROFIBUS-DP-Systems, das von einem Rechner mit eingebautem PROFIBUS Master (CP 5613/CP 5614) gesteuert wird.

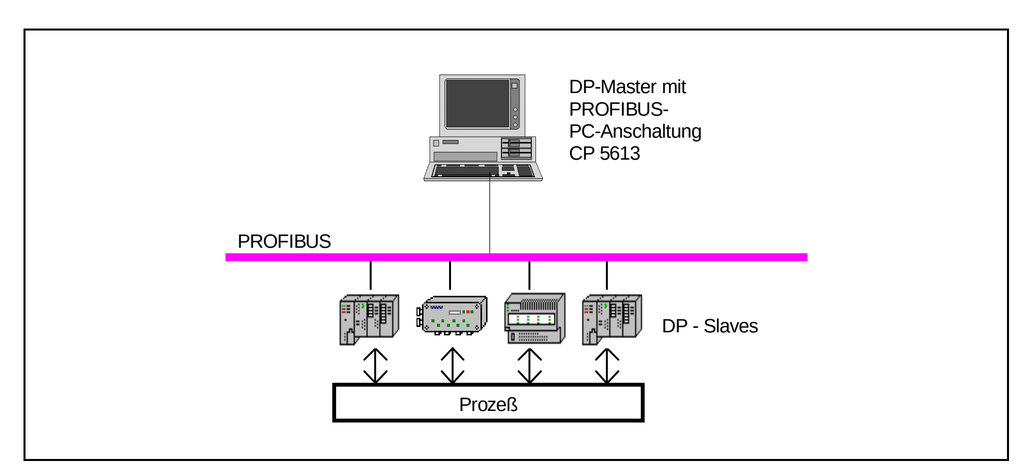



Abbildung 1 Prinzipieller Aufbau einer DP-Anlage

## 2.3 Zyklisches Pollen durch den DP-Master

### Polling

Die Kommunikation zwischen dem DP-Master und den dezentralen Peripheriestationen erfolgt durch Polling. Dies bedeutet, dass der DP-Master in der Produktivphase zyklisch Aufruftelegramme an die ihm zugeordneten DP-Slaves sendet. Für jeden DP-Slave wird ein eigenes Aufruftelegramm gesendet.

In einem Poll-Zyklus werden alle betriebsbereiten DP-Slaves adressiert. Der Adressierung des letzten Slave schließt sich sofort ein weiterer Poll-Zyklus an. Durch dieses Verfahren wird die Aktualität der Daten gewährleistet.

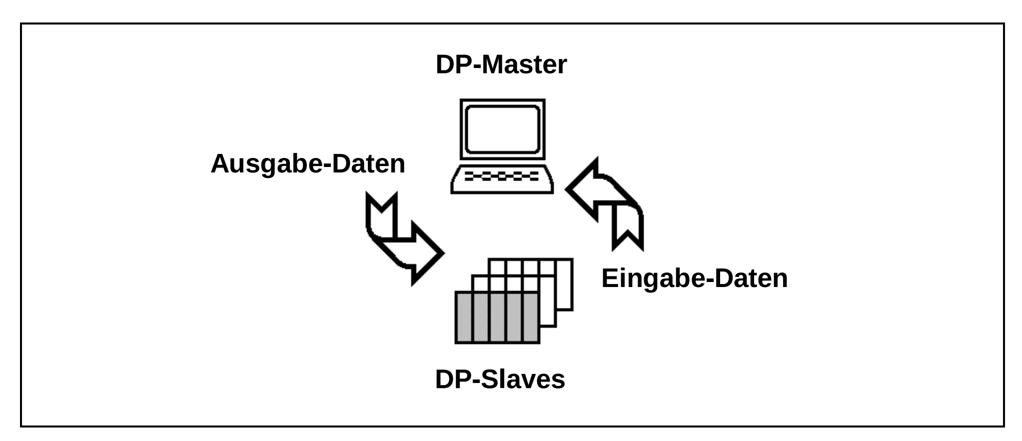



Abbildung 2 Schematische Darstellung des Polling

### Ausgabedaten

Das Aufruftelegramm enthält die aktuellen Ausgabedaten, die der DP-Slave an seinen Ausgabeports anlegen soll. Die Daten dieses Bereichs werden vom DP-Anwenderprogramm vorgegeben. Besitzt ein DP-Slave keine Ausgabe-Ports, wird ihm statt dessen ein „Leertelegramm“ gesendet.

### Eingabedaten

Der Empfang eines Aufruftelegramms muss vom adressierten DP-Slave durch Zurücksenden eines Quittungstelegramms bestätigt werden. Das Quittungstelegramm enthält die aktuellen Eingabedaten, die an den Eingabe-Ports des DP-Slave anliegen. Besitzt ein DP-Slave keine Eingabe-Ports, wird statt dessen ein „Leertelegramm“ zurückgesendet.

## 2.4 Prozessabbild des DP-Masters

### Automatische Aktualisierung der Daten

Der CP 5613 bzw. 5614 als PROFIBUS DP-Master pollt die Daten der Slaves laufend ab und hält sie auf dem PC bereit. Das DP-Anwenderprogramm greift auf diese Daten direkt zu. Dadurch wird das DP-Anwenderprogramm von der Aufgabe des Pollings entlastet.

Abbildung 3 zeigt eine schematische Übersicht der Datenbereiche im DP-Master.

## Datenbereiche

Im DP-Master sind für jeden projektierten DP-Slave drei unterschiedliche Datenbereiche vorhanden:

- Eingabedaten vom DP-Slave
- Ausgabedaten an den DP-Slave
- Diagnosedaten vom DP-Slave

Die Daten werden vom CP 5613 bzw. CP 5614 in einem Speicher gehalten, auf den das DP-Anwenderprogramm direkt - ohne den Umweg über Funktionsaufrufe - zugreifen kann.

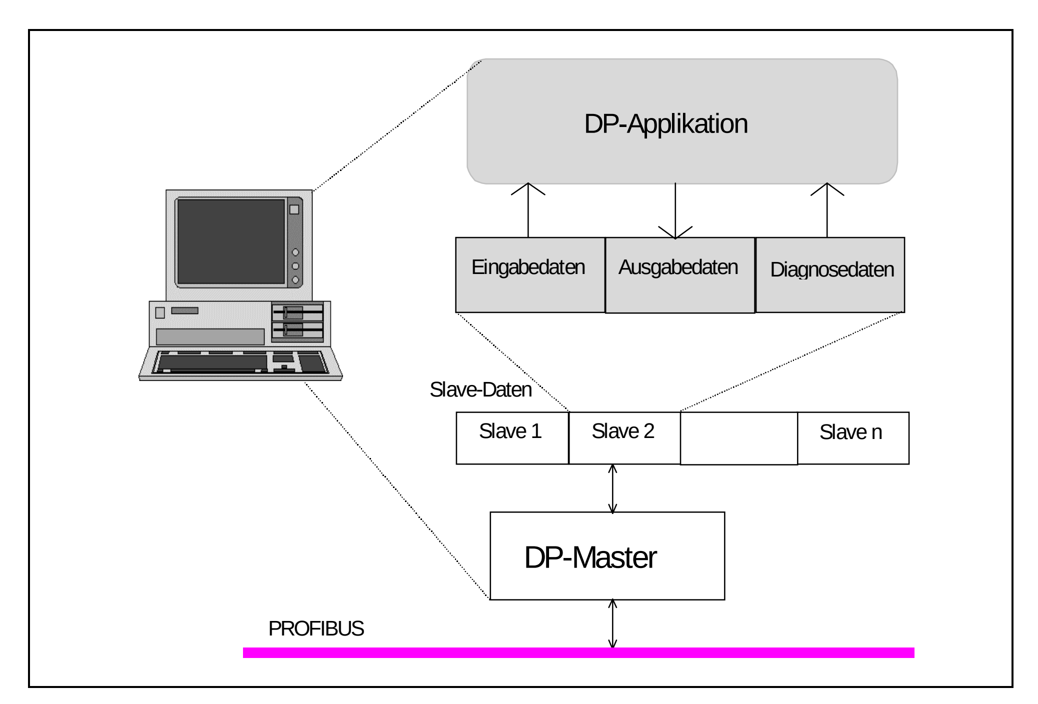



Abbildung 3 Datenbereiche des DP-Masters

## 2.5 Anlauf und Betriebsphase einer DP-Anlage

### Aufgaben des DP-Masters beim Anlauf und im Betrieb

Der DP-Master übernimmt folgende Aufgaben:

- Initialisierung des DP-Systems
- Parametrierung/Konfigurierung der DP-Slave
- zyklischer Datentransfer zu den DP-Slaves
- Überwachen der DP-Slaves
- Bereitstellen von Diagnoseinformationen

### Initialisierung

Der DP-Master kann nur dann Produktivverkehr mit den DP-Slaves durchführen, wenn er diese zuvor parametriert und konfiguriert hat. Die Parametrierung/Konfigurierung erfolgt:

- in der Anlaufphase des DP-Masters.
- nach einem zeitweiligen Ausfall eines DP-Slave während der Produktivphase.

### Parametrierung

Das Parametriertelegramm stellt dem DP-Slave globale Betriebsparameter (z. B. ID) zur Verfügung.

### Konfiguration

Das Konfiguriertelegramm wird nach erfolgreicher Parametrierung des DP-Slave gesendet. Es enthält die aktuelle Konfiguration des DP-Slave. Die Konfiguration enthält die Zahl und die Art der Ein-/Ausgabe-Ports. Der DP-Slave vergleicht das empfangene Konfigurationstelegramm mit den eigenen Werten, die er in der Anlaufphase ermittelt hat. Stimmen die Werte überein, bestätigt der DP-Slave die Konfigurierung und wechselt in die Produktivphase über.

### Betriebszustände

In der Produktivphase wertet der DP-Master die empfangenen Quittungstelegramme der DP-Slaves aus. Aus ihnen kann der DP-Master den aktuellen Betriebszustand der DP-Slaves ermitteln.

## Diagnose

Erkennt ein DP-Slave in der Initialisierungs- oder Produktivphase einen Fehler, kann er diesen in Form von Diagnosedaten an den DP-Master melden. Die empfangenen Diagnosedaten werden im Diagnosebereich des DP-Masters abgelegt. Das DP-Anwenderprogramm muss in diesem Fall die Fehlerreaktion übernehmen.

## 2.6 Die Zustände des DP-Masters

### Überblick

Bei der Kommunikation mit den DP-Slaves kann der DP-Master folgende vier Zustände einnehmen:

- OFFLINE
- STOP
- CLEAR (bzw. AUTOCLEAR)
- OPERATE

### Zustände

Jeder dieser Zustände ist durch definierte Aktionen zwischen DP-Master und den DP-Slaves gekennzeichnet.

| **Master-Zustand** | **Bedeutung** |
|---|---|
| OFFLINE | Es findet keinerlei Kommunikation zwischen DP-Master und den DP-Slaves statt. Dies ist der Startzustand des DP-Masters. |
| STOP | Auch in diesem Zustand findet keinerlei Kommunikation zwischen DP-Master und den DP-Slaves statt. Im Gegensatz zum Zustand OFFLINE kann eine DP-Diagnosestation (DP-Master Klasse 2) Diagnoseinformationen des DP-Masters auslesen. |
| CLEAR (bzw. AUTO-CLEAR) | In diesem Zustand erfolgt die Parametrierung und Konfigurierung aller DP-Slaves, die in der Datenbasis eingetragen und aktiviert sind. Anschließend beginnt der zyklische Datenaustausch zwischen DP-Master und DP-Slaves. Dabei wird im Zustand CLEAR an alle Slaves mit Prozessausgabe der Wert 0 oder Leertelegramme gesendet, d. h. die Prozessausgabe ist deaktiviert. Die Eingabedaten der Slaves sind bekannt und können ausgelesen werden. |
| OPERATE | Im Zustand OPERATE findet der zyklische Produktivdatentransfer zu den DP-Slaves statt. Dies ist die Produktivphase. In diesem Zustand werden reihum die DP-Slaves vom DP-Master angesprochen. Im Aufruftelegramm werden die aktuellen Ausgabedaten, im zugehörigen Antworttelegramm werden die aktuellen Eingabedaten transferiert. |

## Einstellen des Betriebszustands

Beim Anlauf des CP 5613 bzw. CP 5614 durchläuft die Baugruppe, gesteuert vom Anwenderprogramm, die Zustände OFFLINE -> STOP -> CLEAR -> OPERATE.

## 2.7 Entkopplung Slave-Daten und Anwenderprogramm

### Das Prozessabbild ist vom Anwenderprogramm und den Slaves entkoppelt

Aus Gründen der Effizienz sieht die DP-Norm keine Flusskontrolle vor. Die zyklische Aktualisierung des Prozessabbilds der DP-Master-Implementierung ist zeitlich weder an die DP-Slaves noch an das Anwenderprogramm vollständig gekoppelt. Dazu nachfolgend einige Beispiele.

### Beispiel: Slave schreibt zu schnell

Wenn z. B. ein analoger Slave sehr schnell seine Ausgabedaten ändert (Sequenz 1.11, 1.2, 1.3, 1.42, 1.5, 1.6, 1.7, 1.8, ...), dann bekommt der DP-Master beim zyklischen Pollen nur eine Folge von „Schnappschüssen“ mit, die er ins Prozessabbild einträgt (Sequenz 1.11, 1.3, 1.5, 1.8, ...).

### Beispiel: Anwenderprogramm liest zu schnell

Wenn das Anwenderprogramm das Prozessabbild sehr schnell pollt, bekommt es die Werte mehrfach mit, weil es den Poll-Zyklus des Masters überholt. Nach obigem Beispiel liest das Anwenderprogramm dann eine Sequenz 1.11, 1.11, 1.11, 1.11, 1.2, 1.2, 1.2, 1.3, 1.3, 1.3, 1.42, 1.42 ...

### Beispiel: Anwenderprogramm schreibt zu schnell

Wenn das Anwenderprogramm das Prozessabbild sehr schnell ändert (Sequenz 1, 2, 3, 4, 5, 6, ...), überholt es den Poll-Zyklus des DP-Masters. Der überträgt dann nur Schnappschüsse an den Slave (Sequenz 1, 4, 6, ...).

### Beispiel: Anwenderprogramm liest zu langsam

Wenn das Anwenderprogramm das Prozessabbild nur gelegentlich pollt, weil es z. B. in der Zwischenzeit andere Aufgaben auszuführen hat, kann es vorkommen, dass einige Werte übersprungen werden. Aus einer Sequenz 1.1, 1.2, 1.3, 1.4 im Prozessabbild wird dann z. B. 1.1, 1.3 usw. im Anwenderprogramm.

## Abhilfe

Für Anwenderprogramme, die eine stärkere Kopplung mit den Slaves benötigen als oben beschrieben, gibt es eine Reihe von zusätzlichen Möglichkeiten:

- Mit den Hardware-Events des CP 5613 und CP 5614 kann das Anwenderprogramm sich von Änderungen beim Slave benachrichtigen lassen.
- Mit der DP-Protokollerweiterung DP-V1 Master Klasse 1 (DPC1) kann das Anwenderprogramm Daten mit Quittung lesen und schreiben sowie Alarme quittieren (soweit die verwendeten Slaves das unterstützen).
- Durch anwenderspezifische Implementierungen auf Seiten des Anwenderprogramms und der Slaves können - verpackt ins DP-Prozessabbild oder DPC1-Daten - beliebige anwenderspezifische Flusskontrollen zur Kopplung von Master und Slaves erreicht werden, um Datenverluste zu vermeiden.
- Der Hardware-Event bei Zyklusbeginn und Zyklusende des CP 5613/CP 5614 kann zur Synchronisation verwendet werden. Diese Events werden nur im Äquidistanzmodus unterstützt. Die Parametrierung der Äquidistanzparameter ist nur mit STEP7/NCM möglich.

# 2.8 Sicherheitseigenschaften von DP

## Mehrstufiges Sicherheitskonzept

Die DP Programmierschnittstelle bietet ein mehrstufiges Sicherungskonzept, um die Auswirkungen eines Ausfalls der Kommunikationsverbindung oder des DP-Masters zu begrenzen.

- Eine projektierbare Ansprechüberwachung des DP-Slave sorgt dafür, dass ein längere Zeit nicht angesprochener DP-Slave selbständig in einen sicheren Betriebszustand übergeht.
- Eine aktivierbare AUTOCLEAR-Funktion sorgt dafür, dass bei einzelnen nicht ansprechbaren DP-Slaves der DP-Master automatisch in den CLEAR-Zustand übergeht.
- Eine im DP-Master zuschaltbare Aktivitätsüberwachung erkennt die Inaktivität eines DP-Anwenderprogramms und kann so die DP-Slaves in einen sicheren Betriebszustand bringen (ab welcher Software-Version diese Eigenschaft verfügbar ist, können Sie in der Versionstabelle der Installationsanleitung nachlesen).

## Details zur AUTOCLEAR-Eigenschaft

Bei der Projektierung kann die Option AUTOCLEAR eingestellt werden. Tritt dann während der Produktivphase bei einem oder mehreren DP-Slaves ein Fehler auf, wechselt der DP-Master **selbsttätig** in den Zustand AUTOCLEAR über (Herunterfahren des DP Systems). Der Zustand AUTOCLEAR ist gleichbedeutend mit CLEAR. Der DP-Master sendet dann in Ausgaberichtung Daten mit dem Wert 0 oder Leertelegramme an die DP-Slaves. Der Zustand wird vom DP-Master nicht mehr selbständig verlassen, d. h. ein erneuter Übergang in den Zustand OPERATE muss vom Anwender explizit angestoßen werden.

## 2.9 Steuertelegramme an einen oder mehrere Slaves

### Zweck von Steuertelegrammen

Ein Steuertelegramm ist ein Telegramm, das der Master an einen einzelnen Slave, eine oder mehrere Gruppen oder an alle Slaves sendet. Diese Telegramme werden von den angesprochenen Slaves nicht quittiert.

Steuertelegramme dienen der Übertragung von Steuerkommandos (sogenannte Global Controls Commands) an die ausgewählten Slaves zum Zwecke der Synchronisation. Dabei enthält ein Steuerkommando drei Komponenten:

- Kennung, ob ein oder mehrere DP-Slaves adressiert werden
- Identifikation der Slave-Gruppe
- Steuerkommando(s)

### Gruppenbildung

Bei der Projektierung kann einem Slave eine Gruppenidentifikation zugewiesen werden, d. h. es ist möglich, mehrere Slaves in einer Gruppe zusammenzufassen.

Welche der Slaves zu einer Gruppe gehören, wird beim Erstellen der Datenbasis festgelegt. Dabei kann optional für jeden DP-Slave eine Gruppennummer vergeben werden. Diese Gruppennummer wird dem DP-Slave in der Parametrierphase bekannt gemacht. Insgesamt können bis zu acht Gruppen gebildet werden.

## Steuerkommandos

Folgende Steuerkommandos können an DP-Slaves gesendet werden, wenn im konkreten Anwendungsfall erforderlich:

| Steuer kommandos | Beschreibung |
|---|---|
| FREEZE | Nach Erhalt eines FREEZE-Kommandos frieren die DP-Slaves die aktuellen Zustände aller Eingänge ein. Beim nächsten Lesezyklus erhält der DP-Master eine Momentaufnahme der Eingangsdaten. |
| UNFREEZE | Das Einfrieren der Eingänge wird aufgehoben. Die DP-Slaves stellen die aktuellen Eingangsdaten dem DP-Master wieder zur Verfügung. |
| SYNC | Nach Erhalt eines SYNC-Kommandos frieren die DP-Slaves die aktuellen Zustände aller Ausgänge ein. Die danach vom DP-Master gesendeten Ausgangsdaten werden zunächst nicht von den DP-Slaves übernommen. Erst beim nächsten UNSYNC-Kommando werden die Daten übernommen. |
| UNSYNC | Nach Erhalt eines UNSYNC-Kommandos werden die Ausgangsdaten wieder übernommen. |

## 2.10 Typische Sequenzen bei DP

### Grundsätzlicher Ablauf beim DP-Master

Ein DP-Master durchläuft typischerweise folgende Kommunikationssequenz, angestoßen vom Anwenderprogramm:

| Schritt | Bedeutung |
|---|---|
| 1 | Ausgangssituation: Der DP-Master ist im Zustand OFFLINE. |
| 2 | Der DP-Master wechselt in den Zustand STOP. |
| 3 | Der DP-Master wechselt in den Zustand CLEAR. Dabei parametriert und konfiguriert er automatisch die Slaves und beginnt, ihnen zyklisch Nulltelegramme oder leere Telegramme zu schicken (gemäß Projektierung). |
| 4 | Der DP-Master wechselt in den Zustand OPERATE. |
| 5 | Nun werden die Ausgabedaten des Anwenderprogramms zyklisch an die Slaves übertragen, die Eingabedaten gehen von den Slaves zum Anwenderprogramm. |
| 6 | Der Master wird über die Zwischenzustände CLEAR und STOP in den Endzustand OFFLINE geführt und ausgeschaltet. |

### Ausfall eines Slave

Wenn der DP-Master im CLEAR- oder OPERATE-Zustand ist, kann ein Slave ausfallen (d. h. nicht mehr antworten). Der Master versucht dann automatisch, ihn wieder neu zu parametrieren und zu konfigurieren und so wieder in den Zyklus aufzunehmen.

### Aktivieren/Deaktivieren von Slaves

Wenn der DP-Master im CLEAR- oder OPERATE-Zustand ist, können einzelne Slaves aktiviert oder deaktiviert werden. Ein deaktivierter Slave wird vom Master nicht mehr angesprochen.

### AUTOCLEAR

Unter bestimmten Umständen kann der DP-Master vom Zustand OPERATE in den Zustand AUTOCLEAR übergehen; siehe Kapitel 2.8.

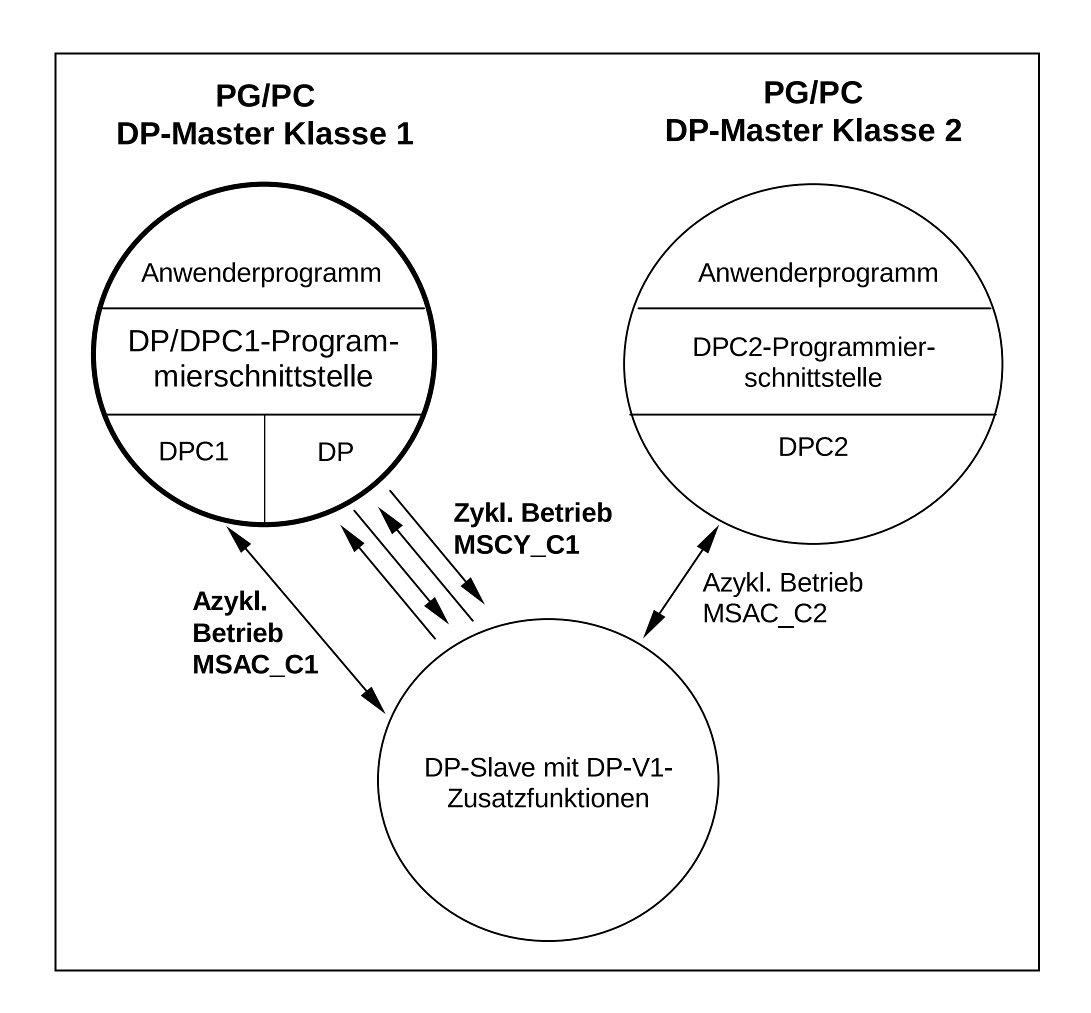
## Empfang von Diagnosedaten

Durch das Zurückgeben von hochprioren Eingabedaten an den Master signalisiert der Slave, dass er Diagnosedaten hat. Der Master holt sie daraufhin ab und stellt sie dem Anwenderprogramm zur Verfügung.

# 2.11 DP-V1 als Erweiterung von DP

## Übersicht DP Protokoll mit DP-V1-Erweiterungen

Neben dem zyklischen DP-Master-Betrieb (siehe Kapitel 2.3) werden als DP-V1 noch zwei Erweiterungen definiert: DPC1 und DPC2. Dazu folgende Übersicht:




## DP-V1 Master Klasse 1 (DPC1)

Ein zyklischer DP-Master kann mittels DPC1 zusätzlich Daten an den Slave schik-ken bzw. lesen, Alarme empfangen und quittieren. Bei den Daten handelt es sich nicht um Prozessdaten, sondern um Slave-spezifische Zusatzdaten (z. B. Nachpa-rametrierung). Diese Daten werden nicht zyklisch gesendet und müssen vom Slave explizit quittiert werden.

## DP-V1 Master Klasse 2 (DPC2)

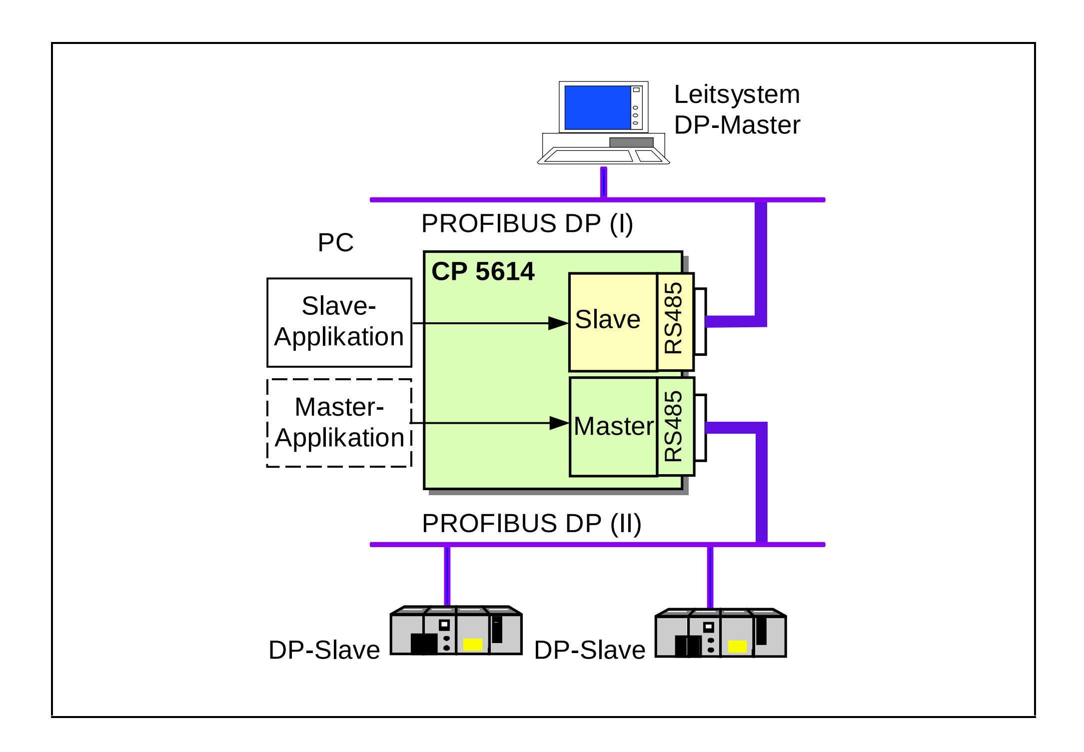
Ein zusätzlicher, nicht zyklisch arbeitender DP-Master kann mittels DPC2 Verbindungen zu Slaves aufbauen und Daten an den Slave schicken bzw. lesen, z. B. für Umparametrierungen oder Diagnose.

Ab welcher Software-Version die DPC2-Funktionen für den CP 5613/CP 5614 angeboten werden, können Sie in der Versionstabelle im Kapitel 14.2 der Installationsanleitung nachlesen.

## 2.12 Slave-Funktionalität des CP 5614

### Die Slave-Funktionalität (nur CP 5614)

Die Piggy-Back-Baugruppe, die mit einem zweiten PROFIBUS-Anschluss auf dem CP 5614 sitzt, besitzt Slave-Funktionalität. Der Slave wird von einem weiteren DP-Master gesteuert.

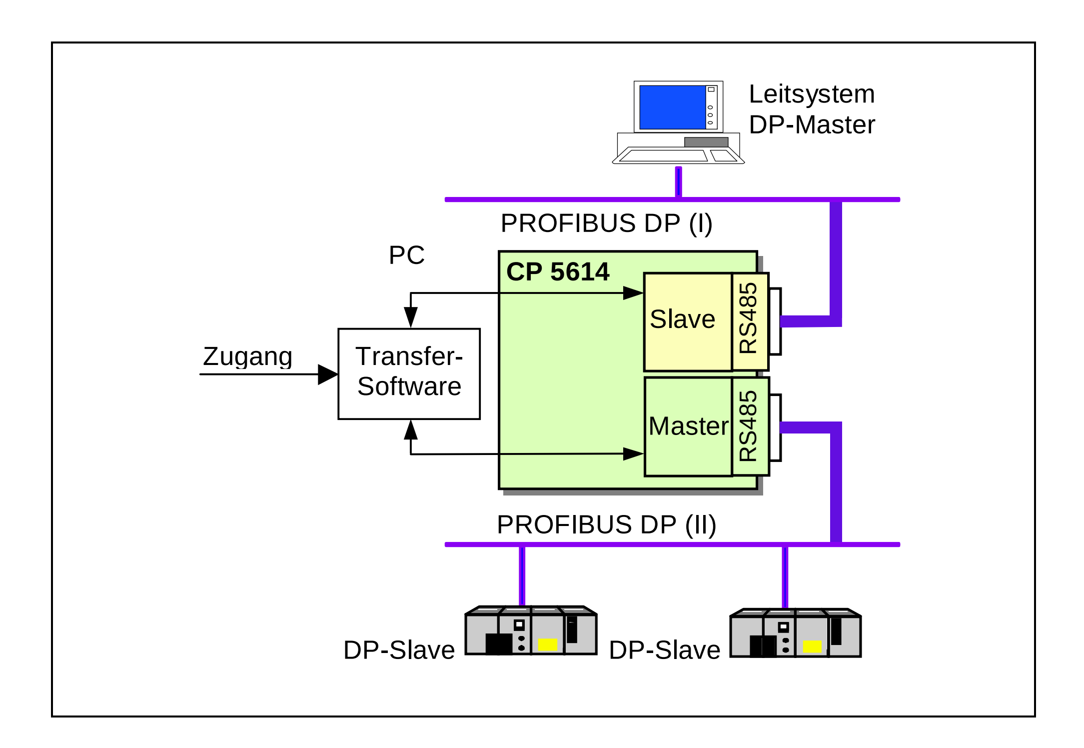


## Die Transfer-Software (nur CP 5614)

Um CP 5614 mit Master- und Slave-Funktionalität zu betreiben, liegt als Beispiel die Transfer-Software vor. Die Transfer-Software überträgt Daten zwischen dem Master- und dem Slave-Teil des CP 5614.

Die Transfer-Software enthält einen Zugang, an dem zusätzlich Eingabe-, Ausgabe- oder Diagnoseaufträge ausgeführt werden können. Ihr Anwenderprogramm kann diesen Zugang benutzen, um zusätzliche Funktionalität dem Transferprogramm hinzuzufügen. (Im Beispielprogramm wird ein Zähler im CP 5614-Slave-Modul hochgezählt.)

Anhand des Beispiels soll verdeutlicht werden, wie eine autonome Transferfunktionalität mit einer lokalen Anwendung gemeinsam auf das Prozessabbild des CP 5614 bzw. CP 5613 zugreifen kann.




## Die Projektierung zur Transfer-Software (nur CP 5614)

Um die autonome Transferfunktionalität festzulegen, wird ebenfalls als Beispiel ein Konfigurationswerkzeug und eine konfigurierte Transferdatei mitgeliefert.

Mit dem Konfigurationswerkzeug lässt sich die Abbildungsfunktionalität also das Kopieren von Daten von Master zu Slave und umgekehrt beliebig festlegen.

# Übersicht DP-Base-Schnittstelle

# 3

Die Programmierschnittstelle des CP 5613/CP 5614 heißt DP-Base-Schnittstelle. In diesem Kapitel lernen Sie als Vorbereitung zur Erstellung eigener DP-Anwendungen die grundsätzlichen Eigenschaften der DP-Base-Schnittstelle kennen, einschließlich typischer Aufruf- und Zugriffsreihenfolgen.

Für eine detaillierte Beschreibung der Funktionsaufrufe und Datenzugriffe lesen Sie bitte im Kapitel 3.10 nach.

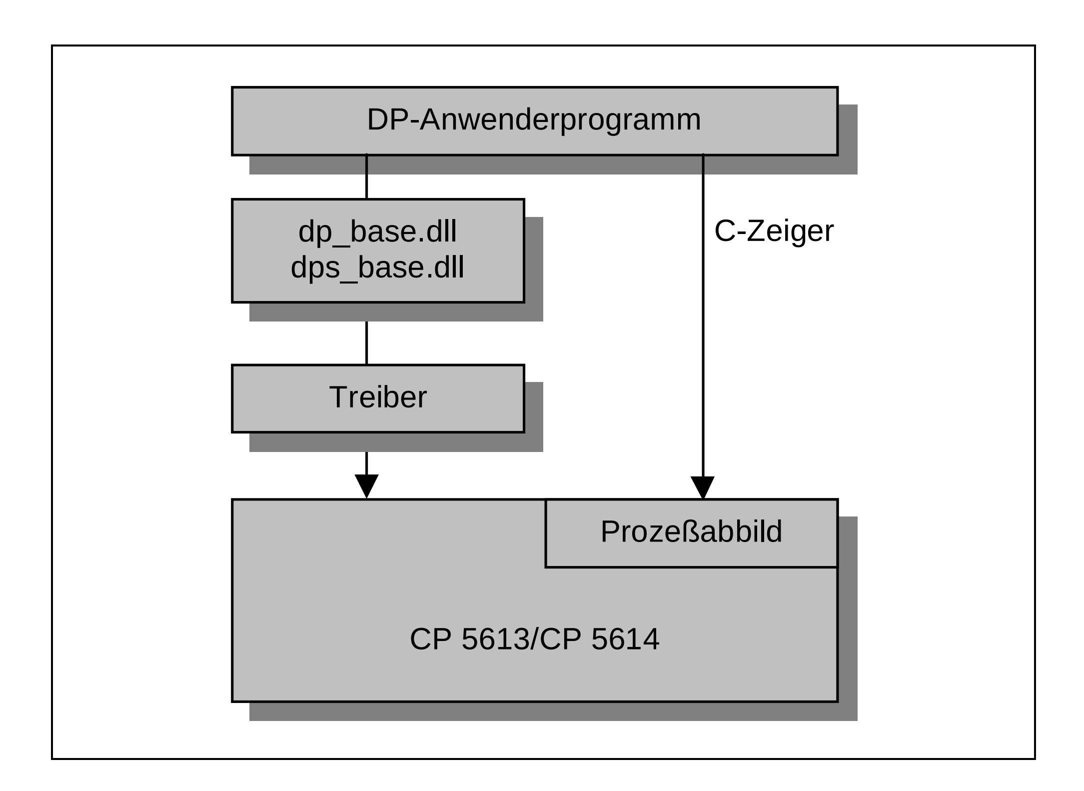
# 3.1 Aufteilung in Funktionen und Daten

## Grundstruktur der DP-Base-Schnittstelle

Die Schnittstelle zum Anwenderprogramm erfolgt über zwei Mechanismen:

- Schnittstellenaufrufe der dp_base.dll bzw. dps_base.dll
- Direktzugriffe auf das Prozessabbild im CP 5613/CP 5614

Dazu folgende Übersicht:




## Funktionen der dp_base.dll und dps_base.dll

Administrative Aufgaben und zur Laufzeit seltener benötigte Kommunikationseigenschaften des CP 5613 und CP 5614 erreicht Ihr Anwenderprogramm durch Aufrufe von Funktionen in den DP-DLLs dp_base.dll und dps_base.dll.

Funktionen, die mit „DP_" beginnen enthalten allgemeine und Master-Modulfunktionalität. Funktionen, die nur für das Slave-Modul relevant sind, beginnen mit „DPS_" (das S steht für Slave-Modul).

## Direktzugriff auf das Prozessabbild

Zur Laufzeit Ihres Anwenderprogramms regelmäßig benötigte Daten des CP 5613 und CP 5614 stehen direkt in einem Speicherbereich des CP zur Verfügung. Darin sind vor allem die Eingabe-, Ausgabe- und Diagnosedaten der DP-Slaves enthalten, aber auch Zustands- und Konfigurationsdaten. Ihr Anwenderprogramm kann mit einem C-Zeiger direkt auf das Prozessabbild zugreifen.

## 3.2 Rolle der Projektierung

### Nutzen der DP-Datenbasis

In der DP-Datenbasis werden Informationen über die Busparameter des PROFIBUS sowie über die im Netz enthaltenen Slaves abgelegt und beim Anlauf durch den CP 5613/CP 5614 berücksichtigt. Dadurch wird Ihr Anwenderprogramm von vielen Details entlastet und muss nicht bei allen Änderungen (z. B. der Datenübertragungsgeschwindigkeit) angepasst werden.

### Projektierung der Slave-Datenbereiche

Bei der Projektierung wird für jeden Slave die Anzahl und Art (Eingabe, Ausgabe, analog, digital) seiner Datenbereiche festgelegt. Diese Konfigurierdaten werden im Anlauf an den Slave geschickt und von diesem geprüft. Stimmen sie nicht mit den tatsächlichen Eigenschaften des Slave überein, kennzeichnet der Slave dies in den Diagnosedaten und wird nicht in den zyklischen Betrieb aufgenommen.

### Aktivierung der Ansprechüberwachung

Ist die Ansprechüberwachung eines DP-Slave in der Projektierung aktiviert, muss der DP-Master innerhalb der vorgegebenen Zeit mit dem DP-Slave kommunizieren.

Geschieht dies nicht, so schaltet der Slave seine Ausgänge in einen sicheren Zustand und nimmt nicht mehr am Datentransfer mit dem Master teil, da der Slave davon ausgeht, dass ein schwerwiegender Fehler aufgetreten ist, zum Beispiel Leitungsbruch oder Ausfall des DP-Masters.

Der Master muss den Slave dann erneut parametrieren und konfigurieren. Erst dann ist wieder der Austausch von Produktivdaten möglich.

Welchen Wert die Ausgänge annehmen, ist aus der Beschreibung der DP-Slaves zu entnehmen.

### Projektierung der AUTOCLEAR-Eigenschaft

Nimmt einer der aktivierten Slaves nicht am Datentransfer teil, und ist zusätzlich die Funktionalität AUTOCLEAR eingestellt, so wechselt der DP-Master automatisch in den Zustand CLEAR (mit der Kodierung AUTOCLEAR).

## Projektierung des „Min_Slave_Interval“

Die Zeit „Min_Slave_Interval“ ist die minimale Zeit zwischen zweimaligem Ansprechen eines Slave durch den Master. Sie wird vom Projektierungswerkzeug anhand der GSD-Daten der Slaves automatisch errechnet.

## 3.3 Konsistenter Zugriff auf das Prozessabbild

### Zugriffskonflikte im Prozessabbild

Wenn Ihr Anwenderprogramm z. B. gerade Daten eines DP-Slave aus dem Prozessabbild liest und in gleichen Augenblick der DP-Master dieses mit neuen Daten überschreibt, könnte Ihr Programm die ersten paar Bytes vom vorigen DP-Zyklus und die letzten Bytes vom aktuellen Zyklus bekommen. Damit wären die Daten verfälscht und inkonsistent.

### Konsistenz beim Lesen wählbar

Beim CP 5613 und CP 5614 können Sie beim Zugriff auf das Prozessabbild wählen, ob Sie Daten konsistent lesen wollen oder nicht. Der Verzicht auf Konsistenz kann z. B. sinnvoll sein, wenn die Daten nur 2 Byte lang sind, denn Inkonsistenz kann erst bei längeren Daten auftreten.

Die Konsistenzsicherung funktioniert über die maximale Datenlänge von 244 Bytes.

### Konsistenz beim Schreiben immer

Beim CP 5613 und CP 5614 ist das Schreiben von Ausgabedaten aufgrund des Übertragungsmechanismus beim CP 5613/CP 5614 immer konsistent, und zwar bis zur maximalen Datenlänge von 244 Bytes.

## 3.4 Mit Hardware-Events arbeiten

### Entlastung der PC-CPU

Um den PC vom Rechenaufwand durch dauerndes Polling an der DP-Schnittstelle zu entlasten, können Sie Hardware-Events einsetzen. Dabei spezifiziert Ihr Anwenderprogramm, bei welchen Ereignissen es vom CP 5613/CP 5614 benachrichtigt werden soll.

### Mögliche Hardware-Events

Als Auslösekriterien für Hardware-Events stehen Ihnen folgende Möglichkeiten zur Verfügung:

- Die Eingabedaten eines DP-Slave haben sich geändert. Der Hardware-Event ist für jeden Slave getrennt aktivierbar.
- Ein DP-Slave schickt Diagnosedaten (egal ob geändert oder nicht). Der Hardware-Event ist für jeden Slave getrennt aktivierbar.
- Fast Logic (siehe Kapitel 3.5)
- Ein neuer DP-Zyklus beginnt
- Der zyklische Teil des DP-Zyklus endet

Beachten Sie bitte, dass die Hardware-Events bei Zyklusanfang und Zyklusende nur im Äquidistanzmodus unterstützt werden. Die Parametrierung der Äquidistanzparameter ist nur mit STEP7/NCM möglich.

### Wie Hardware-Events aktiviert werden

Im Prozessabbildspeicherbereich des CP 5613 und CP 5614 befindet sich ein Steuerbereich für die Aktivierung von Hardware-Events (zur Aktivierung von Fast Logic siehe Kapitel 3.5). Ihr Anwenderprogramm kann sie dort durch einfaches Schreiben in den Speicher setzen und löschen.

## Wie Hardware-Events übermittelt werden

Ein Hardware-Event wird durch Weiterschalten eines Semaphors übermittelt. Damit kann Ihr Anwenderprogramm oder einer dessen Threads auf einzelne Ereignisse warten; siehe hierzu Kapitel 3.6.

---

**Hinweis**

Der Einsatz von Hardware-Events für viele aktive Slaves gleichzeitig kann den PC stärker belasten als Polling; siehe Ratschläge in der FAQ-Liste.

---

## 3.5 Fast Logic

### Zweck

Durch die Fast-Logic-Eigenschaft des CP 5613/CP 5614 können Sie den CP so parametrieren, dass er automatisch Daten von Slaves überwacht und Reaktionen bei anderen Slaves auslöst.

Das hat folgende Vorteile:

- Das Anwenderprogramm wird entlastet.
- Die Datenübertragung findet durch Entkopplung von der PC-Software schneller statt.
- Durch Unabhängigkeit von der PC-Software ist die Reaktion auf das Eingangs-signal garantiert.

Ab welcher Software-Version diese Eigenschaft verfügbar ist, können Sie in der Versionstabelle im Kapitel 14.2 der Installationsanleitung nachlesen.

### Vorgehensweise

Ihr Programm kann Funktionen aufrufen, die Fast Logic parametrieren (DP_fast_logic_on) oder Parametrierungen wieder löschen (DP_fast_logic_off).

## 3.6 Übersicht zum Auslösen und Empfang von Events

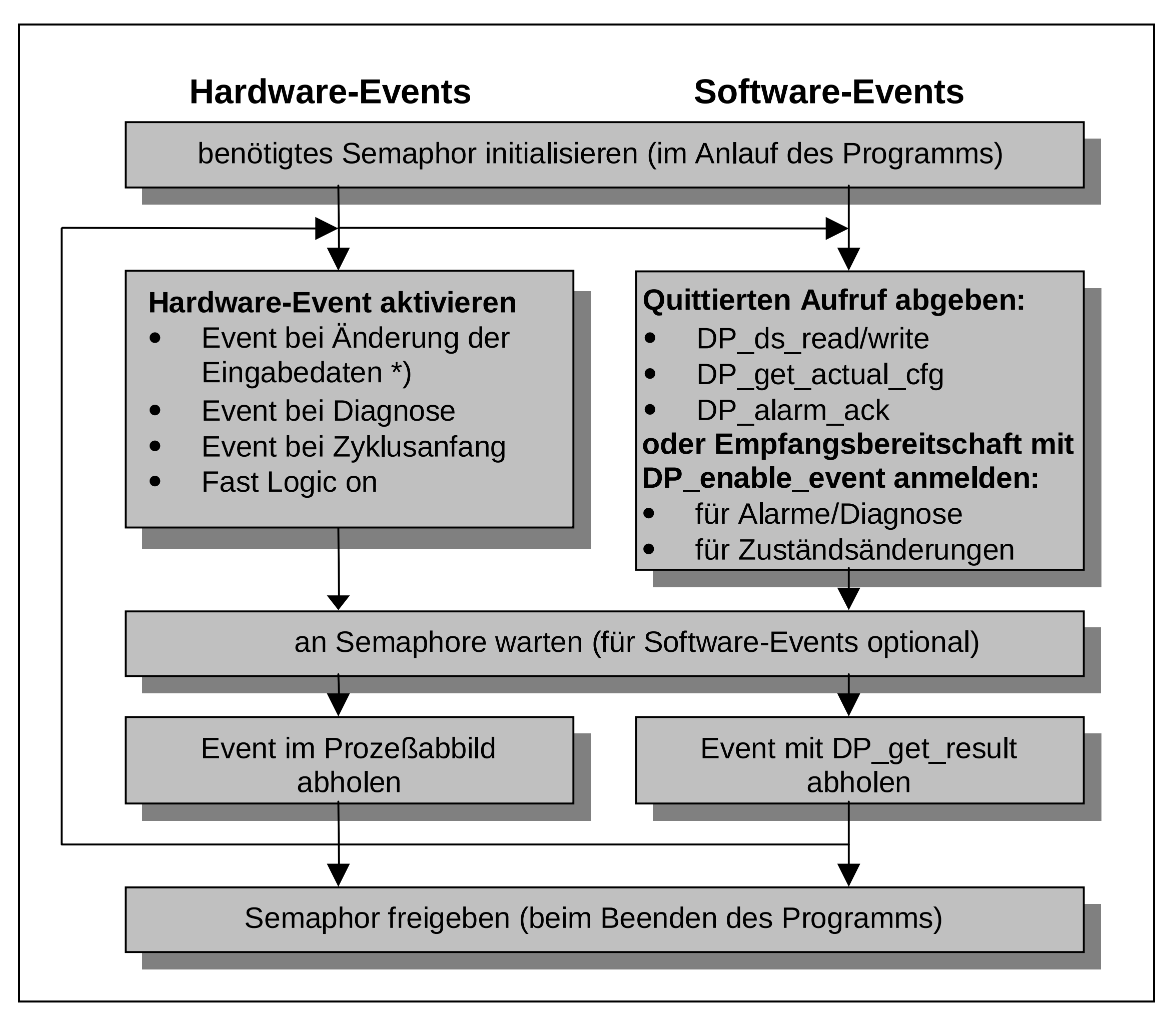
### Eigenschaften von Hardware-Events

Der CP 5613/CP 5614 unterstützt Hardware-Events (siehe Kapitel 3.4) mit Hardware-Mechanismen des CP, so dass sie sehr schnell bearbeitet werden. Hardware-Events werden (bis auf Fast Logic) im Prozessabbild des CP aktiviert, über Semaphore gemeldet und die Details zum eingetroffenen Event liegen im Prozessabbild.

### Eigenschaften von Software-Events

Im Gegensatz dazu werden Software-Events durch Funktionsaufrufe ausgelöst, können mit Semaphore gemeldet werden und werden durch Funktionsaufrufe wieder abgeholt.

### Übersicht zum Ablauf von Events




*) Auch bei Datenänderung im Slave-Modul des CP 5614 möglich.

# 3.7 Typische Sequenzen

## 3.7.1 Initialisierung und Abschluss des Master-Betriebs

### Initialisierung

Die typische Initialisierung eines CP 5613 oder CP 5614 aktiviert den CP und fährt den DP-Master in den OPERATE-Zustand. Dazu sind folgende Schritte erforderlich:

| Schritt | Aktion | Bedeutung |
|---|---|---|
| 1 | DP_start_cp | CP wird initialisiert. |
| 2 | DP_open | Anmelden beim CP. |
| 3 | DP_get_pointer | Exklusiver Zugriff auf das Prozessabbild. |
| 4 | DP_set_mode(Stop) | Master in Zustand STOP fahren. |
| 5 | DP_set_mode(Clear) | Master in Zustand CLEAR fahren, Slaves werden in zyklischen Betrieb genommen, gemäß den Informationen in der Datenbasis. |
| 6 | DP_set_mode(Operate) | Master in Zustand OPERATE fahren. |

### Produktivbetrieb

Das Anwenderprogramm kann beliebig auf die Daten im Prozessabbild zugreifen, DP-V1-Aufträge anstoßen und deren Quittungen abholen sowie sonstige DP-Funktionen anstoßen. Auf den nachfolgenden Seiten werden die hierzu gehörigen Sequenzen näher erläutert.

## Abschluss

Das Herunterfahren des CP fährt den DP-Master in den OFFLINE-Zustand und endet mit dem Anhalten des CP:

| Schritt | Aktion | Bedeutung |
|---|---|---|
| 1 | DP_set_mode(Clear) | Master in Zustand CLEAR fahren, Slaves werden zurückgesetzt |
| 2 | DP_set_mode(Stop) | Master in Zustand STOP fahren, zyklischer Betrieb endet |
| 3 | DP_set_mode(Offline) | Master in Zustand OFFLINE fahren |
| 4 | DP_release_pointer | Zugriff auf das Prozessabbild freigeben |
| 5 | DP_close | Abmelden |
| 6 | DP_reset_cp | CP anhalten |

### 3.7.2 Typische Sequenzen beim pollenden Master-Betrieb

#### Einordnung

Nach dem Initialisieren des CP wie oben beschrieben kann das Anwenderprogramm den CP durch Pollen benutzen, d. h. durch dauernden direkten Zugriff ohne Wartemechanismen.
Ein Zyklus mit Lesen und Schreiben der Prozessdaten kann dabei mit den nachfolgend beschriebenen Mitteln realisiert werden.

## Elemente eines Poll-Zyklus

Alle nachfolgend beschriebenen Schritte erfolgen durch direkten Zugriff auf das Prozessabbild, über den C-Zeiger als Resultat des Aufrufs „DP_get_pointer“.

Zusammengenommen ergeben sie einen beispielhaften Poll-Zyklus.

| Schritt | Aktion | Bedeutung |
|---|---|---|
| 1 | Zustand des Master im Prozessabbild prüfen (USIF_state, Kap. 4.3.6) | Eingabedaten sind nur in den Zuständen OPERATE und CLEAR gültig. Ausgabedaten können nur im Zustand OPERATE gesendet werden. |
| 2 | Zustand der Slaves im Prozessabbild prüfen (slave_state, Kap. 4.3.5) | Die Kommunikation funktioniert nur mit Slaves im Zustand READY. |
| 3 | Optional: Prüfen ob ein Slave Datenänderungen hat (req_mask, Kap. 4.3.4), falls ja: req_mask wieder zurücksetzen | Ihr Anwenderprogramm kann feststellen, ob geänderte Eingabedaten von einem Slave vorliegen. |
| 4 | Eingabedaten der Slaves lesen (slave_in[ ].data, Kap. 4.3.1), Konsistenz durch Zugriff auf D_lock_in_slave_adr | Zur Weiterbearbeitung im Anwenderprogramm |
| 5 | Auf neue Diagnosedaten der Slaves prüfen (diag_count, Kap. 4.3.2) | Wenn der Diagnosezähler sich seit dem letzten Zyklus geändert hat, liegen neue Diagnosedaten vor. |
| 6 | ggf. Diagnosedaten der Slaves lesen (slave_diag[ ].data, Kap. 4.3.2), Konsistenz durch Zugriff auf D_lock_diag_slave_adr | Zur Weiterbearbeitung im Anwenderprogramm |
| 7 | Ausgabedaten der Slaves schreiben (slave_out[ ].data) | Als Resultat der Bearbeitung von Eingabe- und Diagnosedaten |

### 3.7.3 Typische Sequenzen beim pollenden DPC1-Master-Betrieb

#### Einordnung

Nach dem Initialisieren des CP wie oben beschrieben kann das Anwenderprogramm parallel zum zyklischen Betrieb DPC1-Funktionen benutzen, um Daten mit Slaves auszutauschen und Alarme zu beantworten.

Hier wird beschrieben, wie diese Dienste pollend benutzt werden können, d. h. durch dauerndes Abfragen ohne Wartemechanismen.

Einzelne Paare aus Auftrag und Quittung können parallel zueinander verwendet werden.

#### Pollendes Schreiben an einen Slave mittels DPC1

| Schritt | Aktion | Bedeutung |
|---|---|---|
| 1 | Schreibauftrag abgeben (DP_ds_write) | Nach dem Ende der Funktion ist der Auftrag in Bearbeitung. |
| 2 | Quittung abpollen (DP_get_result), bis der Auftrag beendet ist | Die Quittung kann anhand der Order_id im Request-Block wiedererkannt werden. |

#### Pollendes Lesen von einem Slave mittels DPC1

| Schritt | Aktion | Bedeutung |
|---|---|---|
| 1 | Leseauftrag abgeben (DP_ds_read) | Nach dem Ende der Funktion ist der Auftrag in Bearbeitung. |
| 2 | Quittung mit Daten abpollen (DP_get_result), bis der Auftrag beendet ist | Die Quittung kann anhand der Order_id im Request-Block wiedererkannt werden. |

## Empfangen und Beantworten eines DPC1-Alarms

| Schritt | Aktion | Bedeutung |
|---|---|---|
| 1 | Versuch, Alarm zu empfangen (DP_fetch_alarm) | Falls keiner vorliegt, wird das durch das Funktionsergebnis angezeigt. |
| 2 | Falls empfangen: Alarmquittung abgeben (DP_alarm_ack) | Nach dem Ende der Funktion ist der Auftrag in Bearbeitung. |
| 3 | Endequittung abpollen (DP_get_result), bis der Auftrag beendet ist | Die Quittung kann anhand der Order_id im Request-Block wiedererkannt werden. |

## 3.7.4 Typische Sequenzen beim Master-Betrieb mit Hardware-Events

### Einordnung

Nach dem Initialisieren des CP, wie oben beschrieben, kann das Anwenderprogramm Hardware-Events aktivieren und auf deren Eintreffen mit Semaphore warten.

Nach neuen Daten oder nach Diagnose kann Pollen dadurch entfallen und eine Synchronisierung mit dem Zyklusanfang oder dem Zyklusende kann realisiert werden. Diese beiden Events werden nur im Äquidistanzmodus unterstützt. Die Parametrierung der Äquidistanzparameter ist nur mit STEP7/NCM möglich.
Diese Betriebsart kann das in Kapitel 3.7.2 beschriebene Polling ersetzen oder ergänzen.

Nachfolgend wird die Initialisierung dieser Betriebsweise, die Elemente des Dauerbetriebs und das Abmelden erläutert.

### Initialisierung der Semaphore

Vor der ersten Benutzung von Hardware-Events müssen Semaphore wie folgt angelegt werden:

| Schritt | Aktion | Bedeutung |
|---|---|---|
| 1 | Semaphor für Hardware-Events abholen (Funktion DP_init_sema_object) | Die DP-Base-DLL bietet hierzu Semaphore für Änderung der Eingabedaten, Empfang von Diagnose, Zyklusanfang, Zyklusende und Fast Logic an. Der Selektor für Datenänderung ist z. B. DP_OBJECT_TYPE_INPUT_CHANGE. |

## Benutzen von Hardware-Events

Nach dem Abholen der benötigten Semaphore kann folgende Sequenz zum Aktivieren und Abholen von Events durchlaufen werden:

| Schritt | Aktion | Bedeutung |
|---|---|---|
| 1 | Optional: Hardware-Event für Änderung der Eingabedaten einschalten (req_mask, Kap. 4.3.12) | Das Anwenderprogramm erklärt, dass es eine Semaphorweiterschaltung bei Änderung der Eingabedaten haben möchte. |
| 2 | Optional: Hardware-Event für Diagnosedaten einschalten (req_mask, Kap. 4.3.12) | Das Anwenderprogramm erklärt, dass es eine Semaphorweiterschaltung beim Eintreffen von Diagnosedaten haben möchte. |
| 3 | Optional: Hardware-Event für Zyklusanfang einschalten (D_cycle_start_mask, Kap. 4.3.12) | Das Anwenderprogramm erklärt, dass es eine Semaphorweiterschaltung beim Zyklusanfang haben möchte.<br>Dieser Event wird nur im Äquidistanzmodus unterstützt. |
| 4 | Optional: Hardware-Event für Zyklusende einschalten (D_cycle_end_mask, Kap. 4.3.12) | Das Anwenderprogramm erklärt, dass es eine Semaphorweiterschaltung beim Zyklusende haben möchte.<br>Dieser Event wird nur im Äquidistanzmodus unterstützt. |
| 5 | Optional:<br>DP_fast_logic_on | Fast-Logic-Auftrag abgeben |
| 6 | Auf Semaphore warten (z. B. WaitForMultipleObjects) | Das Anwenderprogramm bzw. der aufrufende Thread wartet, bis eines der Ereignisse eintritt.<br>„WaitForMultipleObjects“ ist eine Windows-32-Bit-API-Funktion. |
| 7 | Art des Ereignisses feststellen | Das durchlaufende Semaphor identifiziert die Art des Ereignisses, z. B. Datenänderung. |
| 8 | Quelle des Ereignisses feststellen (welcher Slave) | Durchgehen und prüfen der Flags im Prozessabbild:<br>• req_mask = DPR_DATA_CHANGE bei Datenänderung (Kap. 4.3.4)<br>• diag_count geändert bei Diagnose (Kap. 4.3.2) |
| 9 | Ereignis lesen und bearbeiten | • Lesen durch Zugriff auf das Prozessabbild:<br>— slave_in[n].data bei Eingabedaten<br>— slave_diag[n].data bei Diagnose<br>— siehe Kap. 4.3.10 für Fast Logic<br>• Weiterreichen an andere Teile des Anwenderprogramms. |

## Abmeldung der Semaphore

Nach der letzten Benutzung von Hardware-Events melden Sie Ihre Semaphore wie folgt ab:

| Schritt | Aktion | Bedeutung |
|---|---|---|
| 1 | Semaphor für Events abmelden (Funktion DP_delete_sema_object) | Gibt das vorher abgeholte Semaphor wieder frei. |

## 3.7.5 Typische Sequenzen beim DPC1-Betrieb mit Semaphore

### Einordnung

Die im Kapitel 3.7.3 beschriebene pollende Betriebsweise für DPC1 kann auch durch einen Betrieb mit Semaphoren ersetzt werden.
Nachfolgend werden die Initialisierung dieser Betriebsweise, die Elemente des Dauerbetriebs und das Abmelden erläutert.

### Initialisierung der Semaphore

Vor der ersten Benutzung muss das Semaphor wie folgt angelegt werden:

| Schritt | Aktion | Bedeutung |
|---|---|---|
| 1 | Semaphor für asynchrone Aufträge abholen (Funktion DP_init_sema_object) | Die DP-Base-DLL bietet hierzu ein Semaphor für alle asynchronen Aufträge (Typ DP_OBJECT_TYPE_ASYNC). |

### Benutzen von Semaphore für DPC1

| Schritt | Aktion | Bedeutung |
|---|---|---|
| 1 | Schreibauftrag abgeben (DP_ds_write) | Nach dem Ende der Funktion ist der Auftrag in Bearbeitung. |
| 2 | Auf Semaphor warten (z. B. WaitForMultipleObjects) | Das Anwenderprogramm bzw. der aufrufende Thread wartet, bis das Ereignis eintritt. „WaitForMultipleObjects“ ist eine Windows-32-Bit-API-Funktion. |
| 3 | Quittung abholen (DP_get_result) | Die Quittung kann anhand der Order_id im Request-Block wiedererkannt werden. |

Die anderen DPC1-Dienste funktionieren analog.

## Abmeldung eines Semaphor

Nach der letzten Benutzung melden Sie Ihr Semaphor wie folgt ab:

| Schritt | Aktion | Bedeutung |
|---|---|---|
| 1 | Semaphor für Events ab-melden (Funktion DP_delete_sema_object) | Gibt das vorher abgeholte Semaphor wieder frei. |

## 3.8 Eigenschaften des CP 5614 (Slave-Funktionen, Transfer-Software)

### Zusammenspiel zwischen Master- und Slave-Funktionalität

Der DP-Master und der Slave auf dem CP 5614 können sowohl parallel als auch einzeln betrieben werden.

### Betriebsart „einfacher Slave“

In dieser Betriebsart werden dem CP beim Aufruf der Funktion DPS_open (... slave_mode=DPS_SM_SIMPLE ...) alle Daten übergeben, die nötig sind, um das Slave-Modul in den Datenaustausch zu bringen. Das hat den Vorteil, dass das Anwenderprogramm einfach Ausgänge lesen und Eingänge schreiben kann, ohne sich um den Zustand des Slave-Moduls zu kümmern, bzw. ohne die Parameter- und Konfigurationsdaten zu überprüfen.

### Betriebsart „dynamischer Slave“

In dieser Betriebsart kann der Slave sich dynamisch auf die Projektierung seines Masters einstellen. Beim DPS_open() wird nicht „DPS_SM_SIMPLE“ gewählt. Dem Anwender wird mitgeteilt, wenn Parameter- und Konfigurationsdaten empfangen werden. Er prüft dann, ob er die Einstellungen akzeptiert. Das ermöglicht eine größere Flexibilität, vor allem, wenn der Slave modular aufgebaut ist. In diesem Fall kann die Sicht des Masters mit der tatsächlichen Konfiguration des Slave verglichen werden.

### Transfer-Software

Im Lieferumfang ist eine Beispiel-Software enthalten, mit der Daten zwischen verschiedenen Slaves des Master-Teils und dem Slave-Modul rangiert werden können. Das ist vor allem vorteilhaft für Anwendungsfälle, in denen mit dem Master-Teil ein unterlagerter Bus gesteuert wird, und der PC über das Slave-Modul z. B. mit einem Leitrechner, verbunden ist.

# 3.9 Typische Sequenzen für das Slave-Modul CP 5614

## 3.9.1 Initialisierung und Abschluss des Slave-Moduls im „Simple“-Modus

### Initialisierung

Die Initialisierung des CP 5614 im „Simple“-Modus aktiviert den CP und initialisiert das Slave-Modul, so dass es vom Master in den Produktivzustand gebracht werden kann. Dazu sind folgende Schritte erforderlich:

| Schritt | Aktion | Bedeutung |
|---|---|---|
| 1 | DP_start_cp | CP wird initialisiert. |
| 2 | DPS_open | Initialisieren des Slave-Moduls -<br>Im Parameter „slave_mode“ ist das Bit „DPS_SM_SIMPLE“ gesetzt, im Parameter „init_data“ stehen die erwarteten Parametrier- und Konfigurierdaten. |
| 3 | DPS_start | Starten des Slave-Moduls |
| 4 | DP_get_pointer | Exklusiver Zugriff auf das Prozessabbild |

### Produktivbetrieb

Das Anwenderprogramm kann beliebig auf die Daten im Prozessabbild lesend und schreibend zugreifen.

### Abschluss

Zum Herunterfahren des CP fährt das Slave-Modul in den OFFLINE-Zustand und endet mit dem Anhalten des CP:

| Schritt | Aktion | Bedeutung |
|---|---|---|
| 1 | DPS_stop | Slave in den OFFLINE-Zustand bringen |
| 2 | DP_release_pointer | Zugriff auf das Prozessabbild freigeben |
| 3 | DPS_close | Abmelden beim Slave-Modul |
| 4 | DP_reset_cp | CP anhalten |

## 3.9.2 Initialisierung und Abschluss des Slave-Moduls im „Dynamic“-Modus

### Initialisierung

Die Initialisierung des CP 5614 im „Dynamic“-Modus aktiviert den CP und initialisiert das Slave-Modul, so dass es am Bus antwortet. Das vom Master gesendete Parameter- und Konfigurationstelegramm muss vom Anwender überprüft und quittiert werden. Dazu sind folgende Schritte erforderlich:

| Schritt | Aktion | Bedeutung |
|---|---|---|
| 1 | DP_start_cp | CP wird initialisiert |
| 2 | DPS_open | Initialisieren des Slave-Moduls:<br>im Parameter „slave_mode“ ist das Bit „DPS_SM_SIMPLE“ **nicht** gesetzt, im Parameter „init_data“ steht die Default-Konfiguration |
| 3 | DPS_start | Starten des Slave-Moduls |
| 4 | DP_get_pointer | Exklusiver Zugriff auf das Prozessabbild |

## Pollender Produktivbetrieb

Das Anwenderprogramm kann beliebig auf die Daten im Prozessabbild zugreifen, muss jedoch darauf gefasst sein, dass Parameter- und Konfig-Telegramme bearbeitet werden müssen. Das ist dann der Fall, wenn der Master das Slave-Modul in den Datenaustausch bringen will (z. B. beim Anlauf des Masters, nach Ziehen und Stecken des Bussteckers, ...). Die folgende Sequenz muss in einer Programmhauptschleife zyklisch abgearbeitet werden.

| Schritt | Aktion | Bedeutung |
|---|---|---|
| 1 | DPS_get_ind | Nachfragen, ob Indications eingetroffen sind. |
| 2a | Wenn DPS_CHK_PRM:<br>DPS_set_resp | Falls eine neue Parametrierung eingetroffen ist:<br>• User-Parameterdaten prüfen und<br>• positiv oder negativ quittieren |
| 2b | Wenn DPS_CHK_CFG:<br>DPS_set_resp | Falls eine neue Konfiguration eingetroffen ist:<br>• Konfigurationsdaten prüfen und<br>• positiv oder negativ quittieren. |
| 3 | Zugriff auf PA<br>(PA - **Prozessabbild**) | Das Lesen und Schreiben von Daten im Prozessabbild (beliebiger Zugriff) |
| 4 | Zu Schritt 1 gehen. | DPS_get_ind muss zyklisch aufgerufen werden. |

**Hinweis**

Nach dem positiven Quittieren der Konfigurationsdaten müssen die Eingabedaten im Prozessabbild des Slave-Moduls mindestens einmal geschrieben werden (Initialisierung), bevor der Slave-Modul in den Datenaustausch gehen kann.

## Abschluss

Beim Herunterfahren des CP fährt das Slave-Modul in den OFFLINE-Zustand und der CP wird angehalten:

| Schritt | Aktion | Bedeutung |
|---|---|---|
| 1 | DPS_stop | Slave in den OFFLINE-Zustand bringen |
| 2 | DP_release_pointer | Zugriff auf das Prozessabbild freigeben |
| 3 | DPS_close | Abmelden beim Slave-Modul |
| 4 | DP_reset_cp | CP anhalten |

## 3.9.3 Typische Sequenzen mit Semaphore beim Slave-Modul

### Einordnung

Die im Kapitel 3.7.3 beschriebene pollende Betriebsweise kann auch durch einen Betrieb mit Semaphore ersetzt werden. Nachfolgend werden die Initialisierung dieser Betriebsweise, die Elemente des Dauerbetriebs und das Abmelden erläutert.

### Initialisierung des Semaphors

Vor der ersten Benutzung muss das Semaphor wie folgt angelegt werden:

| Schritt | Aktion | Bedeutung |
|---|---|---|
| 1 | Semaphor für asynchrone Aufträge abholen (Funktion DP_init_sema_object) | Die DP-Base-Lib bietet hierzu einen Semaphor für alle asynchronen Aufträge (Typ DP_OBJECT_TYPE_ASYNC). |

### Benutzen von Semaphore für das Slave-Modul

| Schritt | Aktion | Bedeutung |
|---|---|---|
| 1 | Auf Semaphor warten (z. B. WaitForMultipleObjects) | Das Anwenderprogramm bzw. der aufrufende Thread wartet, bis ein Ereignis eintritt. „WaitForMultipleObjects“ ist eine Windows-32-Bit-API-Funktion. |
| 2 | DPS_get_ind | Angekommene Indications abholen. |
| 3a | Wenn DPS_CHK_PRM:<br>DPS_set_resp | Falls eine neue Parametrierung eingetroffen ist:<br>User-Parameterdaten prüfen, und positiv oder negativ quittieren. |
| 3b | Wenn DPS_CHK_CFG:<br>DPS_set_resp | Falls eine neue Konfiguration eingetroffen ist:<br>Konfigurationsdaten prüfen, und positiv oder negativ quittieren. |
| 4 | Zu Schritt 1 gehen | |

**Hinweis**

Nach dem positiven Quittieren der Konfigurationsdaten müssen die Eingabedaten im Prozessabbild des Slave-Moduls mindestens einmal geschrieben werden (Initialisierung), bevor der Slave-Modul in den Datenaustausch gehen kann.

## Abmeldung des Semaphors

Nach der letzten Benutzung melden Sie Ihr Semaphor wie folgt ab:

| Schritt | Aktion | Bedeutung |
|---|---|---|
| 1 | Semaphor für Events abmelden (Funktion DP_delete_sema_object) | Gibt das vorher abgeholte Semaphor wieder frei. |

## 3.10 Mehrere Protokolle, Anwenderprogramme, CPUs

### Multi-CP-Betrieb

Ab welcher Software-Version der gleichzeitige Betrieb mehrerer CP 5613/CP 5614 unterstützt wird, können Sie in der Versionstabelle im Kapitel 14.2 der Installationsanleitung nachlesen.

### Mehrere Anwenderprogramme

Ab welcher Software-Version der gleichzeitige Betrieb mehrerer Anwendungsprogramme unterstützt wird, können Sie in der Versionstabelle im Kapitel 14.2 der Installationsanleitung nachlesen.

### Mehrere CPUs in einem PC

Der Betrieb in PCs mit mehreren CPUs wird unterstützt.

# Beschreibung der einzelnen DP-Funktionen und Daten sowie Fehlercodes

# 4

In diesem Kapitel sind die einzelnen Funktionen und Zugriffsmöglichkeiten auf Daten im Prozessabbild des CP 5613 und CP 5614 detailliert beschrieben.

Außerdem erfahren Sie die Bedeutung der möglichen Fehlercodes.

Die Datenformate für E/A-Daten und für Diagnose werden ebenso beschrieben.

Das Kapitel eignet sich vor allem als Nachschlagewerk beim Erstellen Ihrer Anwenderprogramme.

## 4.1 Liste der Funktionen CP 5613 und CP 5614

### Darstellungskonventionen

In den folgenden Funktionsbeschreibungen gibt es in der Funktionsdeklaration generelle Darstellungen, die hier beschrieben werden:

| Darstellung | Bedeutung |
|---|---|
| // in | Der Wert wird vom Anwenderprogramm als Input der Funktion zur Verfügung gestellt. |
| // out | Der Wert wird von der Funktion an das Anwenderprogramm zurückgegeben. |
| // inout | Der Wert muss vorbesetzt werden und ist nach Abschluss der Funktion aktualisiert. |

### Einheitliches Aufrufschema der Funktionen der dp_base.dll

Die Funktionen sind nach einem einheitlichen Schema aufgebaut:

```
Fehlerklasse = DP_... (    Vorgabeparameter 1,
                                  ...
                           Vorgabeparameter n,
                           DP_ERROR_T *error);
```

Jede Funktion gibt als Rückgabewert eine der im folgenden beschriebenen Fehlerklassen zurück. Im Fehlerfall enthält die Struktur DP_ERROR_T – in Abhängigkeit der Fehlerklasse – detaillierte Fehlerinformationen (siehe Kapitel 4.4). Die Fehlerauswertung und die Funktion DP_get_err_txt werden auch in den mitgelieferten Beispielprogrammen erläutert.

In den Funktionsbeschreibungen werden die Return-Werte wie folgt tabellarisch beschrieben, erläutert anhand des Funktionsaufrufs DP_reset_cp:

- Falls der Return-Wert DP_OK ist, so bedeutet dies, dass die Funktion erfolgreich ausgeführt wurde. In diesem Beispiel ist der CP also tatsächlich zurückgesetzt worden.
- Falls der Return-Wert DP_ERROR_CI ist und die Komponente error_code in der Fehlerstruktur DP_ERROR_T den Wert CI_RET_RESET_CP_USED hat, so bedeutet dies, dass der CP nicht zurückgesetzt worden ist, weil noch andere Anwendungen den CP benutzen. Auf solche ausdrücklich angegebenen Fehlercodes sollte Ihr Anwenderprogramm vorbereitet sein.
- In allen anderen Fällen liegt ein Fehler vor, der nur in Ausnahmefällen auftritt.

Eine Liste der Fehlercodes finden Sie in Kapitel 4.4.2 „Fehlercodes“.

## Header-Dateien

Die C-Header-Dateien dp_5613.h und dps_5614.h mit der genauen C-Beschreibung der Funktionen und Datenstrukturen finden Sie im Unterverzeichnis „prog“ Ihrer Software-Installation.

## Synchrone und asynchrone Funktionen

Soweit nicht anders erwähnt, ist die Ausführung der gewünschten Aufgabe mit dem Ende der Funktion abgeschlossen (d. h. synchron).
Einige Funktionen (z. B. DP_ds_read) stoßen hingegen lediglich die Bearbeitung an und beenden sich dann. Der eigentliche Abschluss der Bearbeitung muss dann gesondert durch den Aufruf der Funktion DP_get_result in Erfahrung gebracht werden (d. h. asynchron).

### 4.1.1 Übersichtstabellen zu den Funktionen

#### Administrative Funktionen

| Name | Aufgabe |
|---|---|
| DP_start_cp | Laden der Firmware und der Datenbasis in den CP 5613/CP 5614. |
| DP_open | Anmelden eines DP-Anwenderprogramms, Vergabe eines User-Handle. |
| DP_get_pointer | Zeiger auf das Prozessabbild anfordern. |
| DP_init_sema_object | Diese Funktion richtet ein Semaphor ein, an der Ihr Anwenderprogramm auf das Eintreffen von Ereignissen warten kann. |
| DP_delete_sema_object | Diese Funktion meldet ein Semaphor wieder ab. |
| DP_release_pointer | Zeiger auf das Prozessabbild wieder zurückgeben. |
| DP_close | Mit dieser Funktion meldet sich ein DP-User wieder ab. |
| DP_reset_cp | Anhalten der CP-Firmware. |

#### Standard-DP-Funktionen

| Name | Aufgabe |
|---|---|
| DP_set_mode | Mit dieser Funktion wird die gewünschte DP-Zustand (OFFLINE, STOP, CLEAR, OPERATE) eingestellt. |
| DP_slv_state | Mit dieser Funktion kann ein Slave explizit aktiviert oder deaktiviert sowie die AUTOCLEAR-Eigenschaft verändert werden. |
| DP_read_slv_par | Mit dieser Funktion können die Parameter eines DP-Slave aus der Datenbasis ausgelesen werden. |
| DP_global_ctrl | Mit dieser Funktion kann ein Global Control Command an einen oder mehrere Slaves gesendet werden. |

## Funktionen zum DP-V1 Master Klasse 1 (DPC1)

| Name | Aufgabe |
|---|---|
| DP_ds_read | Diese Funktion sendet einen „Datensatz lesen“–Aufruf an einen DP-V1-Slave. |
| DP_ds_write | Diese Funktion sendet einen „Datensatz schreiben“–Aufruf an einen DP-V1-Slave. |
| DP_fetch_alarm | Diese Funktion holt einen DPC1-Alarm, eine DPC1-Statusmeldung oder sonstige Diagnosedaten ab. |
| | |
| DP_alarm_ack | Diese Funktion sendet einen „Alarm-Acknowledge“-Aufruf an einen DP-V1-Slave. |
| DP_get_cref | Diese Funktion ermittelt aus der Slave-Adresse und der User-Id die Kommunikationsreferenz (c_ref) für DP-V1-Aufträge. |
| DP_get_actual_cfg | Mit dieser Funktion können die aktuellen Konfigurationsdaten eines Slave ausgelesen werden. |

## Hilfsfunktionen

| Name | Aufgabe |
|---|---|
| DP_get_err_txt | Diese Funktion gibt Fehlerinformationen im Klartext aus. |
| DP_enable_event | Diese Funktion aktiviert das Warten auf Zustandsänderungen oder Diagnose bei Slaves. |
| DP_get_result | Mit dieser Funktion wird die Quittung eines asynchronen Aufrufs und andere Software-Events abgeholt. |
| DP_disable_event | Diese Funktion macht DP_enable_event rückgängig |
| DP_watchdog | Diese Funktion startet oder beendet eine Aktivitätsüberwachung der DP-Master-Anwenderprogramm. |
| DP_write_trc | Mit dieser Funktion kann ein DP-Anwenderprogramm eigene Trace-Ausgaben in ein Trace-File schreiben. |

## Fast Logic

| Name | Aufgabe |
|---|---|
| DP_fast_logic_on | Diese Funktion parametriert den CP 5613/CP 5614 zur automatischen Überwachung eines Slave und Datenübertragung an einen anderen Slave. |
| DP_fast_logic_off | Diese Funktion nimmt eine mit DP_fast_logic_on durchgeführte Parametrierung zurück. |

## 4.1.2 DP_start_cp

### Zweck

Mit dieser Funktion wird der CP 5613/CP 5614 initialisiert. Die auf dem CP ablaufende Firmware und die Datenbasis werden heruntergeladen.

### Syntax

```
DPR_DWORD DP_start_cp  (const DPR_STRING    *cp_name,
     // in
                        const DPR_STRING    *database,
     // in
                        DP_ERROR_T     *error );    // out
```

### Parameter

| Name | Beschreibung |
|---|---|
| cp_name | Logischer Name des CP (z. B. „CP_L2_1:“) |
| database | Pfad und Name der Datenbasis – falls ein Nullzeiger angegeben ist, so wird der über das Werkzeug „PG/PC-Schnittstelle einstellen“ eingestellte Datenbasisname verwendet. |
| error | Adresse einer vom Anwenderprogramm bereitgestellten Struktur vom Typ DP_ERROR_T - Die Struktur enthält im Fehlerfall Details zur Fehlerursache (siehe Kapitel 4.4). |

## Return-Wert

| Name | Beschreibung |
|---|---|
| DP_OK | Erfolgreicher Abschluss der Funktion.<br>Die grüne Token-LED leuchtet. |
| DP_ERROR_CI<br>und error->error_code ==<br>CI_RET_START_ALREADY_<br>DATABASE | Fehler: Der CP ist bereits mit einer ande-<br>ren Datenbasis gestartet worden. |
| DP_ERROR_CI<br>und error->error_code ==<br>CI_RET_START_ALREADY_<br>DONE | Fehler: Der CP ist bereits mit derselben<br>Datenbasis gestartet worden.<br>Der Fehler kann ignoriert werden. |
| Andere | Fehlerhafter Abschluss der Funktion. |

## 4.1.3 DP_reset_cp

### Zweck

Mit dieser Funktion wird der CP 5613/CP 5614 rückgesetzt. Danach ist der CP nicht mehr am Bus aktiv (Token-LED ist aus).
Wenn noch andere Anwendungen den CP nutzen, setzt ihn diese Funktion nicht zurück, verwenden Sie dazu ggf. das Konfigurations- und Diagnosewerkzeug „PG/PC-Schnittstelle einstellen“ (über das Startmenü von Windows NT erreichbar).

---

**Hinweis**

Um den CP wieder zu benutzen, muss Ihr Anwenderprogramm als nächstes den Aufruf DP_start_cp verwenden.

---

### Syntax

```
DPR_DWORD DP_reset_cp  (const DPR_STRING *cp_name,   // in
                         DP_ERROR_T       *error );   // out
```

### Parameter

| Name | Beschreibung |
|---|---|
| cp_name | Logischer Name des CP (z. B. „CP_L2_1:“) |
| error | Adresse einer vom Anwenderprogramm bereitgestellten Struktur vom Typ DP_ERROR_T - Die Struktur enthält im Fehlerfall Details zur Fehlerursache (siehe Kapitel 4.4). |

### Return-Wert

| Name | Beschreibung |
|---|---|
| DP_OK | Erfolgreicher Abschluss der Funktion |
| DP_ERROR_CI und error->error_code == CI_RET_RESET_CP_USED | Nicht erfolgreich, weil andere Anwendungen den CP noch nutzen. |
| andere | Fehlerhafter Abschluss der Funktion |

## 4.1.4 DP_open

### Zweck

Mit dieser Funktion meldet sich ein DP-Anwenderprogramm zur Kommunikation an. Im Erfolgsfall gibt die Funktion ein User-Handle zurück. Das User-Handle muss bei allen weiteren Funktionsaufrufen verwendet werden.

### Syntax

```
DPR_DWORD  DP_open (
           const DPR_STRING  *cp_name,         // in
           DPR_DWORD   *user_handle,     // out
           DP_ERROR_T  *error );         // out
```

### Parameter

| Name | Beschreibung |
|---|---|
| cp_name | Logischer Name des CP (z. B. „CP_L2_1:“) |
| user_handle | Zeiger auf User-Handle-Variable - Im Erfolgsfall wird hier das dem Anwenderprogramm zugeordnete User-Handle eingetragen. |
| error | Adresse einer vom Anwenderprogramm bereitgestellten Struktur vom Typ DP_ERROR_T - Die Struktur enthält im Fehlerfall Details zur Fehlerursache (siehe Kapitel 4.4). |

### Return-Wert

| Name | Beschreibung |
|---|---|
| DP_OK | Erfolgreicher Abschluss der Funktion. |
| DP_ERROR_CI und<br>error->error_code ==<br>CI_RET_OPEN_CP_NOT_<br>STARTED | CP nicht gestartet. |
| DP_ERROR_EVENT_NET und<br>error->error_code ==<br>DP_RET_CP USR_NOT_<br>COMPATIBLE | Die dplib.dll und die dp_base.dll können sich nicht beim gleichen CP anmelden. |
| DP_ERROR_RES und<br>error->error_code ==<br>DP_RET_CP_TOO_MANY_USR | Es können sich keine weiteren DP-Instanzen am CP anmelden. |
| DP_ERROR_RES und<br>error->error_code ==<br>DP_RET_CP_NO_DP_PAR | Es sind keine DP-Parameter in der Datenbasis des CP enthalten. |
| DP_ERROR_RES und<br>error->error_code ==<br>DP_RET_TOO_MANY_USR | Es können sich keine weiteren Instanzen an der dp_base.dll anmelden. |
| Andere | Fehlerhafter Abschluss der Funktion. |

### 4.1.5 DP_get_pointer

#### Zweck

Mit dieser Funktion erhält ein DP-Anwenderprogramm exklusiv den Zeiger auf die Prozessdaten des CP 5613/CP 5614. Mit Hilfe dieses Zeigers kann das DP-Anwenderprogramm anschließend direkt auf das Datenabbild des CP 5613/CP 5614 zugreifen.

#### Allgemeine Hinweise

---

**Hinweis 1**

Jeweils nur ein Programm kann zu einer Zeit einen Zeiger auf das Prozessabbild haben. Dadurch werden Zugriffskonflikte auf die Register zur Konsistenzsteuerung im Prozessabbild (siehe z. B. D_lock_in_slave_adr in Kap. 4.3.1) vermieden.

---

---

**Hinweis 2**

Beachten Sie, dass ein Zugriff auf das Dualport RAM ohne gültigen Zeiger unter Windows NT zu einer Schutzverletzung führt. Ursachen dazu sind: DP_get_pointer wurde nicht aufgerufen, DP_get_pointer wird mit einer Fehlermeldung beendet oder der Zeiger wurde mit DP_release_pointer wieder freigegeben.

---

---

**Hinweis 3**

Diese Funktion ist relativ laufzeitintensiv und beeinträchtigt bei häufigem Aufrufen die Echtzeiteigenschaften Ihres Anwenderprogramms.

---

---

**Hinweis 4**

Der CP muss mit DP_start_cp schon gestartet worden sein.

---

## Syntax

```
DPR_DWORD DP_get_pointer (
     DPR_DWORD                     user_handle,     // in
     DPR_DWORD                     timeout,         // in
     DPR_CP5613_DP_T   volatile   **dpr,            // out
     DP_ERROR_T                    *error );        // out
```

## Parameter

| Name | Beschreibung |
|---|---|
| user_handle | User-Handle, das beim Aufruf DP_open vergeben wurde. |
| timeout | Dauer der maximalen Wartezeit (in Millisekunden), bis die Funktion zurückkehrt.<br>**Grenzwerte:**<br>0: keine Wartezeit (Funktion kehrt sofort zurück)<br>0x7FFFFFFE: maximale Wartezeit<br>DP_TIMEOUT_FOREVER: „unendliche“ Wartezeit |
| dpr | Adresse eines vom Anwenderprogramm bereitgestellten Zeigers. Im Erfolgsfall wird hier die Zugriffsadresse auf das Prozessabbild des CP 5613/CP 5614 eingetragen - Mit Hilfe dieses Zeigers kann das DP-Anwenderprogramm anschließend direkt auf das Datenabbild des CP 5613/CP 5614 zugreifen. |
| error | Adresse einer vom Anwenderprogramm bereitgestellten Struktur vom Typ DP_ERROR_T - Die Struktur enthält im Fehlerfall Details zur Fehlerursache (siehe Kapitel 4.4). |

## Return-Wert

| Name | Beschreibung |
|---|---|
| DP_OK | Erfolgreicher Abschluss der Funktion |
| DP_ERROR_EVENT_NET und error->error_code == DP_RET_TIMEOUT | Innerhalb der angegebenen Wartezeit konnte der Zeiger auf die Prozessdaten nicht zur Verfügung gestellt werden.<br>Ursache: ein anderes Programm hatte während dieser Zeit exklusiven Zugriff auf die Prozessdaten. |
| andere | Fehlerhafter Abschluss der Funktion |

## 4.1.6 DP_release_pointer

### Zweck

Mit dieser Funktion gibt ein DP-Anwenderprogramm den Zeiger auf die Prozessdaten wieder zurück. Danach darf das DP-Anwenderprogramm nicht mehr direkt auf das Datenabbild des CP 5613/CP 5614 zugreifen.

---

**Hinweis**

Diese Funktion ist relativ laufzeitintensiv und beeinträchtigt bei häufigem Aufrufen die Echtzeiteigenschaften Ihres Anwenderprogramms.

---

### Syntax

```
DPR_DWORD  DP_release_pointer(
                 DPR_DWORD   user_handle,            // in
                 DP_ERROR_T  *error );               // out
```

### Parameter

| Name | Beschreibung |
|---|---|
| user_handle | User-Handle, das beim Aufruf DP_open vergeben wurde. |
| error | Adresse einer vom Anwenderprogramm bereitgestellten Struktur vom Typ DP_ERROR_T. Die Struktur enthält im Fehlerfall Details zur Fehlerursache (siehe Kapitel 4.4). |

### Return-Wert

| Name | Beschreibung |
|---|---|
| DP_OK | Erfolgreicher Abschluss der Funktion |
| andere | Fehlerhafter Abschluss der Funktion |

## 4.1.7 DP_close

### Zweck

Mit dieser Funktion meldet sich ein DP-Anwenderprogramm wieder von der Kommunikation mit dem CP ab.

---

**Hinweis 1**

Nach dem erfolgreichen Abmelden ist das User-Handle nicht mehr gültig und darf nicht mehr weiter verwendet werden. Ebenso ist auch der Zeiger auf das Dualport RAM, welcher durch die Funktion DP_get_pointer ermittelt wurde, nicht mehr gültig.

---

**Hinweis 2**

Um den DP-Master herunterzufahren, sollte Ihre Anwenderprogramm vorher den Master mit der Funktion DP_set_mode in den Zustand OFFLINE bringen.

---

**Hinweis 3**

Um den CP ganz vom Bus zu trennen, kann Ihre Anwendung anschließend die Funktion DP_reset_cp verwenden.

---

### Syntax

```
DPR_DWORD  DP_close    (DPR_DWORD   user_handle,     // in
                         DP_ERROR_T  *error );        // out
```

### Parameter

| Name | Beschreibung |
|---|---|
| user_handle | User-Handle, das beim Aufruf DP_open vergeben wurde. |
| error | Adresse einer vom Anwenderprogramm bereitgestellten Struktur vom Typ DP_ERROR_T - Die Struktur enthält im Fehlerfall Details zur Fehlerursache (siehe Kapitel 4.4). |

### Return-Wert

| Name | Beschreibung |
|---|---|
| DP_OK | Erfolgreicher Abschluss der Funktion |
| andere | Fehlerhafter Abschluss der Funktion |

## 4.1.8 DP_get_err_txt

### Zweck

Diese Funktion ermittelt aus den Rückgabewerten der Fehlerstruktur DP_ERROR_T detaillierte Fehlerinformationen im Klartext. Eine detaillierte Beschreibung der möglichen Fehlercodes finden Sie im Kapitel 4.4, Fehler.

### Syntax

```
DPR_DWORD  DP_get_err_txt (
           DP_ERROR_T        *error,                 // in
           const DPR_STRING  *language,              // in
           DPR_STRING        text[DP_ERR_TXT_SIZE] ); // out
```

### Parameter

| Name | Beschreibung |
|---|---|
| error | Adresse einer vom Anwenderprogramm bereitgestellten Struktur vom Typ DP_ERROR_T. Die Struktur enthält die Error-Werte einer zuvor aufgerufenen DP-Funktion (siehe Kapitel 4.4). |
| language | Sprache des auszugebenden Fehlertextes:<br>• „German“<br>• „English“ |
| text | Zeiger auf Datenbereich, in welchem der Fehlertext inclusiv abschließendem Null-Byte abgelegt wird. |

### Return-Wert

| Name | Beschreibung |
|---|---|
| DP_OK | Erfolgreicher Abschluss der Funktion |
| andere | Fehlerhafter Abschluss der Funktion |

**Hinweis**

Wird im Vorgabeparameter „language“ ein anderer String als „German“ oder „English“ eingetragen, so wird der Fehlertext in englischer Sprache ausgegeben.

## 4.1.9 DP_set_mode

### Zweck

Mit dieser Funktion wird der gewünschte DP-Zustand (OFFLINE, STOP, CLEAR, OPERATE) eingestellt



**Warnung**

Wenn der Master von DP_CLEAR nach DP_OPERATE wechseln soll, aber AUTOCLEAR (siehe 2.8) projektiert ist und ein oder mehrere Slaves ausgefallen sind, kann er nicht in den Zustand DP_OPERATE wechseln. Er wechselt dann stattdessen nach DP_AUTOCLEAR, was von der Bedeutung her mit DP_CLEAR identisch ist. Ihr Anwenderprogramm darf also nicht endlos pollen, ob der Zustand DP_OPERATE erreicht ist, sondern muss auch prüfen, ob der Master infolge eines Slave-Problems in den Zustand DP_AUTOCLEAR gewechselt hat (Zelle USIF_state, siehe Kap. 4.3.6).

**Hinweis 1**

Da das Erreichen des neuen Betriebszustands längere Zeit dauern kann (abhängig von Datenübertragungsgeschwindigkeit, Zahl der Slaves etc.), wartet die Funktion nicht, bis der neue Betriebszustand erreicht worden ist. Dadurch wird verhindert, dass das Anwenderprogramm unzulässig lang verzögert werden kann.

**Hinweis 2**

Beim Einstellen eines neuen Zustands dürfen keine Zustände übersprungen werden. Ausgehend von der aktuellen Zustand müssen die Zustände in der vorgegebenen (aufsteigenden oder absteigenden) Reihenfolge OFFLINE <-> STOP <-> CLEAR <-> OPERATE durchlaufen werden. Nach einem DP_set_mode-Aufruf muss durch nachfolgende Zugriffe auf den „Master-Info“-Bereich des Prozessabbilds (Zelle USIF_state, siehe Kap. 4.3.6) geprüft werden, ob der neue Zustand erreicht worden ist. Erst danach darf ein neuer Zustand eingestellt werden

**Hinweis 3**

Wenn Ihr DP-Master statt in den Zustand OPERATE in den Zustand AUTOCLEAR ging, weil ein Slave mit AUTOCLEAR-Eigenschaft nicht in den Produktivbetrieb gehen konnte, können Sie den DP-Master trotzdem nach OPERATE bringen. Deaktivieren Sie dazu vorher den Slave oder seine AUTOCLEAR-Eigenschaft (jeweils durch den Aufruf DP_set_slv_state).

## Syntax

```
DPR_DWORD DP_set_mode (DPR_DWORD   user_handle,      // in
                       DPR_WORD    mst_mode,         // in
                       DP_ERROR_T  *error );         // out
```

## Parameter

| Name | Beschreibung |
|---|---|
| user_handle | User-Handle, das beim Aufruf DP_open vergeben wurde. |
| mst_mode | Neuer Betriebszustand, der eingestellt werden soll. Folgende Konstanten können verwendet werden:<br>DP_OFFLINE: OFFLINE-Zustand<br>DP_STOP: STOP-Zustand<br>DP_CLEAR: CLEAR-Zustand<br>DP_OPERATE: OPERATE-Zustand |
| error | Adresse einer vom Anwenderprogramm bereitgestellten Struktur vom Typ DP_ERROR_T. Die Struktur enthält im Fehlerfall Details zur Fehlerursache (siehe Kapitel 4.4). |

## Return-Wert

| Name | Beschreibung |
|---|---|
| DP_OK | Die Änderung des DP-Zustands wurde aktiviert. |
| DP_ERROR_EVENT_NET und error->error_code ==<br>• DP_RET_CP_SET_MODE_CLR_ACT<br>• DP_RET_CP_SET_MODE_OFFL_ACT<br>• DP_RET_CP_SET_MODE_OPR_ACT<br>• DP_RET_CP_SET_MODE_STOP_ACT | Ein DP_set_mode Aufruf ist noch in Bearbeitung. |
| DP_ERROR_EVENT_NET und error->error_code ==<br>• DP_RET_CP_WRONG_MODE_CLR<br>• DP_RET_CP_WRONG_MODE_OFL<br>• DP_RET_CP_WRONG_MODE_OPR<br>• DP_RET_CP_WRONG_MODE_STP | Der gewählte Master-Zustand ist nicht zulässig (Überspringen eines Master-Zustands). |
| Andere | Fehlerhafter Abschluss der Funktion |

### 4.1.10 DP_slv_state

#### Zweck

Mit dieser Funktion kann der Betriebszustand eines DP-Slave während der Laufzeit des DP-Anwenderprogramms geändert werden. Der Slave kann dabei aus der Bearbeitung herausgenommen oder wieder aktiviert werden. Darüber hinaus kann die AUTOCLEAR-Eigenschaft eines Slave verändert werden.

Mit der Slave-Adresse 0xFF und slv_mode == DP_AUTOCLEAR_ON/ DP_AUTOCLEAR_OFF wird die Eigenschaft AUTOCLEAR generell aktiviert/deaktiviert.

Optional können selektiv einzelne Slaves (slv_add != 0xFF) aus der AUTOCLEAR-Bearbeitung hinein- oder herausgenommen werden.

---

**Hinweis 1**

Wurde die AUTOCLEAR-Eigenschaft in der Datenbasis projektiert, ist die AUTOCLEAR-Eigenschaft aller Slaves bereits eingeschaltet.

---

**Hinweis 2**

Wurde die AUTOCLEAR-Eigenschaft in der Datenbasis nicht projektiert, und wollen Sie diese einschalten, so müssen Sie zuerst die AUTOCLEAR-Eigenschaft generell (Slave-Adresse 0xFF) einschalten und dann die AUTOCLEAR-Eigenschaft für die einzelnen Slaves aktivieren.

---

**Hinweis 3**

Die Verwendung der Parameter DP_SLV_ACTIVATE und DP_SLV_DEACTIVATE ist in einer DP-Applikation normalerweise nicht erforderlich, da die Aktivierung oder Deaktivierung eines DP-Slaves automatisch durch den Kommunikationsprozessor erfolgt (abhängig vom Betriebszustand des Masters). Diese Parameter sollten nur in Ausnahmefällen verwendet werden, um einen Slave generell, d.h. unabhängig vom Betriebszustand des DP-Masters, aus dem DP-Zyklus herauszunehmen oder danach wieder hereinzunehmen.

---

**Hinweis 4**

Erfolgt der Aufruf mit dem Parameter DP_SLV_RESTART, so wird der Slave neu initialisiert (heruntergefahren und danach erneut parametriert und konfiguriert). Der Restart ist erforderlich, wenn im Master-Zustand Clear oder Operate einer der asynchronen DPC1-Aufrufe ( DP_ds_read, DP_ds_write, DP_alarm_ack) mit einem Kommunikationsfehler beendet wird oder wenn die Funktion DP_fetch_alarm einen Datenfehler meldet. Die Durchführung des Restarts selbst kann eine gewisse Zeit in Anspruch nehmen. Das Anwenderprogramm kann sich über die Funktion DP_enable_event die Beendigung des Restarts anzeigen lassen. Der Restart ist dann erfolgreich abgeschlossen, wenn die Funktion DP_enable_event für den entsprechenden Slave das Event DP_SLAVE_ENTER meldet.

## Syntax

```
DPR_DWORD  DP_slv_state(DPR_DWORD  user_handle,      // in
                        DPR_WORD   slv_add;          // in
                        DPR_WORD   slv_mode,         // in
                        DP_ERROR_T *error );         // out
```

## Parameter

| Name | Beschreibung |
|---|---|
| user_handle | User-Handle, das beim Aufruf DP_open vergeben wurde. |
| slv_add | Adresse des Slave oder 0xFF mit der Bedeutung:<br>AUTOCLEAR generell ein- bzw. ausschalten. |
| slv_mode | Gewünschter Betriebszustand des Slave:<br>• DP_SLV_ACTIVATE  Slave aktivieren<br>• DP_SLV_DEACTIVATE  Slave deaktivieren<br>• DP_SLV_RESTART  Slave herunterfahren und neu starten<br>• AUTOCLEAR-Einstellung:<br>• DP_AUTOCLEAR_ON  aktivieren<br>• DP_AUTOCLEAR_OFF  deaktivieren<br>Die Werte können mit Oder verknüpft werden. |
| error | Adresse einer vom Anwenderprogramm bereitgestellten Struktur vom Typ DP_ERROR_T. Die Struktur enthält im Fehlerfall Details zur Fehlerursache (siehe Kapitel 4.4). |

### Return-Wert

| Name | Beschreibung |
|---|---|
| DP_OK | Erfolgreicher Abschluss der Funktion |
| DP_ERROR_EVENT_NET und<br>error->error_code ==<br>DP_RET_CP_REQ_ACTIV | Es ist noch ein DP_slv_state-Request in Bearbeitung.<br>Mögliche Reaktion des Anwendungsprogramms: nach einer kurzen Wartezeit erneut DP_slv_state aufrufen. |
| andere | Fehlerhafter Abschluss der Funktion |

### 4.1.11 DP_read_slv_par

#### Zweck

Mit dieser Funktion können die Parameter eines momentan betriebenen DP-Slaves vom CP 5613/CP 5614 erfragt werden.

#### Syntax

```
DPR_DWORD  DP_read_slv_par  (DPR_DWORD   user_handle,// in
                              DPR_WORD    slv_add,    // in
                              DPR_WORD    type,       // in
                              DPR_WORD    *data_len;  // out
                              DPR_BYTE    *data;      // out
                              DP_ERROR_T  *error );   // out
```

#### Parameter

| Name | Beschreibung |
|---|---|
| user_handle | User-Handle, das beim Aufruf DP_open vergeben wurde. |
| slv_add | Adresse des Slave |
| type | Gewünschte Datenart (siehe Kap. 4.7):<br>DP_SLV_TYP: allg. Slave-Parameter<br>DP_SLV_PRM: Parametrierdaten<br>DP_SLV_CFG: Konfiguration<br>DP_SLV_USR: User-spezifische Parametrierdaten<br>Für die letzen beiden werden in der Regel keine Daten geliefert. |
| data_len | Adresse einer Längenvariablen - Hier wird die Zahl der gültigen Bytes im Datenpuffer eingetragen. |
| data | Zeiger auf einen Datenbereich - Der Datenbereich muss mindestens DP_PAR_SIZE groß sein (siehe Kap. 4.7). |
| error | Adresse einer vom Anwenderprogramm bereitgestellten Struktur vom Typ DP_ERROR_T - Die Struktur enthält im Fehlerfall Details zur Fehlerursache (siehe Kapitel 4.4). |

### Return-Wert

| Name | Beschreibung |
|---|---|
| DP_OK | Erfolgreicher Abschluss der Funktion |
| DP_ERROR_EVENT_NET und<br>error->error_code ==<br>DP_RET_CP_ADR_NOT_IN_DB | Fehler: der angegebene Slave existiert nicht in der Datenbasis. |
| andere | Fehlerhafter Abschluss der Funktion |

## 4.1.12 DP_global_ctrl

### Zweck

Mit dieser Funktion kann ein Global Control Command (Steuerkommando) an einen oder mehrere Slaves gesendet werden.

---

**Hinweis**

Die Zuordnung von Slaves zu einer Slave-Gruppe erfolgt fest über die Projektierung der DP-Datenbasis. Ein Slave wertet das Global Control Command nur aus, wenn er einer der angesprochenen Gruppen zugeordnet ist.

---

### Syntax

```
DPR_DWORD  DP_global_ctrl(
                       DPR_DWORD   user_handle,      // in
                       DPR_WORD    slv_add,          // in
                       DPR_BYTE    command,          // in
                       DPR_BYTE    group,            // in
                       DP_ERROR_T  *error);          // out
```

## Parameter

| Name | Beschreibung |
|---|---|
| user_handle | User-Handle, das beim Aufruf DP_open vergeben wurde. |
| slv_add | Adresse des Slave oder Sammeladresse (DP_BROADCAST_ADR) |
| command | Hier können folgende Global Control-Kommandos (auch durch Oder verknüpft) vorgegeben werden:<br>• DP_SYNC: Einfrieren der Ausgabebytes<br>• DP_UNSYNC: Aufheben des Kommandos DP_SYNC<br>• DP_FREEZE: Die Zustände der Eingänge werden eingelesen und eingefroren<br>• DP_UNFREEZE: Aufheben des Kommandos DP_FREEZE |
| group | Die anzusprechenden Slave-Gruppen, wobei Bit 0 die Gruppe 1 bedeutet, Bit 1 die Gruppe 2 usw. Die Bits können mit Oder verknüpft werden. Mindestens ein Bit muss gesetzt sein. |
| error | Adresse einer vom Anwenderprogramm bereitgestellten Struktur vom Typ DP_ERROR_T. Die Struktur enthält im Fehlerfall Details zur Fehlerursache (siehe Kapitel 4.4). |

## Return-Wert

| Name | Beschreibung |
|---|---|
| DP_OK | Erfolgreicher Abschluss der Funktion |
| DP_ERROR_EVENT_NET und error->error_code == DP_RET_CP_ADR_NOT_IN_DB | Slave-Adresse nicht in der Datenbasis. |
| DP_ERROR_EVENT_NET und error->error_code == DP_RET_CP_REQ_NOT_ALLOWED | Der Dienst ist nicht erlaubt. |
| DP_ERROR_EVENT_NET und error->error_code == DP_RET_CP_SLV_NOT_ACTIV | Der betreffende Slave ist inaktiviert. |
| DP_ERROR_EVENT_NET und error->error_code == DP_RET_CP_TOO_MANY_CTRL_CMD | Zu viele Global-Ctrl-Commands sind zur Zeit in Bearbeitung. |
| DP_ERROR_REQ_PAR und error->error_code == DP_RET_CP_WRONG_FREEZE_GRP | Nur bei Einzeladressierung.<br>Der adressierte Slave gehört keiner der angegebenen Gruppen an (Kommando war FREEZE oder UNFREEZE). |
| DP_ERROR_REQ_PAR und error->error_code == DP_RET_CP_WRONG_SYNC_GRP | Nur bei Einzeladressierung.<br>Der adressierte Slave gehört keiner der angegebenen Gruppen an (Kommando war SYNC oder UNSYNC). |
| DP_ERROR_REQ_PAR und error->error_code == DP_RET_CP_WRONG_GC_GRP | Mindestens eine der angegebenen Gruppen ist leer. |
| DP_ERROR_REQ_PAR und error->error_code == DP_RET_CP_WRONG_GC_CMD | Ungültiger Global-Ctrl-Command. |
| Andere | Fehlerhafter Abschluss der Funktion |

### 4.1.13 DP_ds_read

#### Zweck

Diese Funktion sendet einen „Datensatz lesen“ – Aufruf an einen DP-V1-Slave. Das Lesen der DP-V1-Daten erfolgt parallel zum zyklischen Betrieb. Es handelt sich hierbei nicht um den Slave-Input, sondern um ein zusätzliches Datenpaket mit einer Adressierung über Slot und Index eines Slave. Der Aufbau und die Bedeutung der Daten kann je nach Slave-Typ variieren.

---

**Hinweis**

Der Lesevorgang wird **asynchron** ausgeführt (DP_OK_ASYNC). Die Quittung muss mit DP_get_result abgeholt werden.

---

#### Syntax

```
DPR_DWORD  DP_ds_read (DPR_DWORD   user_handle,     // in
                       DPC1_REQ_T        *request,   // in
                       DP_ERROR_T        *error );   // out
```

## Parameter

| Name | Beschreibung |
|---|---|
| user_handle | User-Handle, das beim Aufruf DP_open vergeben wurde. |
| Request | Zeiger auf DP-V1-Struktur DPC1_REQ_S mit den Einträgen für Datensatz-Lesen-Auftrag.<br><br>typedef struct DPC1_REQ_S<br>{<br>DPR_WORD order_id; // in<br>DPR_DWORD c_ref; // in<br>union<br>{<br>DP_DS_READ_T dp_ds_read;<br>DP_DS_WRITE_T dp_ds_write;<br>DP_ALARM_ACK_T dp_alarm_ack;<br>DP_ENABLE_EVT_T dp_enable_evt;<br>DP_GET_CFG_T dp_get_cfg;<br>}req;<br>} DPC1_REQ_T;<br><br>mit<br><br>typedef struct DP_DS_READ_S<br>{<br>DPR_BYTE slot_number; // in<br>DPR_BYTE index; // in<br>DPR_BYTE length_s; // inout<br>DPR_BYTE data_s[DPR_DPC1_MAX_LENGTH];<br>// out<br>} DP_DS_READ_T; |

Fortsetzung der Tabelle auf der nächsten Seite

Fortsetzung der Tabelle von der letzten Seite

| Name | Beschreibung |
|---|---|
| „Request“ Fortsetzung | Order_id ist eine vom Anwenderprogramm frei vergebbare Kennung für den Auftrag. Diese Kennung wird in der asynchronen Quittung unverändert zurückgegeben und kann dafür benützt werden, die Quittung dem Auftrag zuzuordnen.<br>Das Element c_ref spezifiziert den Slave. Über die Hilfsfunktion DP_get_cref(user_id, slv_add) kann der Wert für c_ref ermittelt werden.<br>Zu den Elementen slot_number und index schlagen Sie bitte die Beschreibung des jeweiligen Slave nach.<br>Das Element length_s gibt als Vorgabewert die Länge der zu lesenden Daten an. Nach Empfang der DP_ds_read-Confirmation enthält length_s die tatsächliche Anzahl der vom Slave empfangenen Daten an. Die empfangenen Daten werden in data_s eingetragen. |
| error | Adresse einer vom Anwenderprogramm bereitgestellten Struktur vom Typ DP_ERROR_T. Die Struktur enthält im Fehlerfall Details zur Fehlerursache (siehe Kapitel 4.4). |

## Return-Wert

| Name | Beschreibung |
|---|---|
| DP_OK_ASYNC | Die Funktionsbearbeitung wurde erfolgreich aktiviert. |
| DP_ERROR_EVENT_NET und error->error_code == DP_RET_REQ_ACTIV | Es ist noch ein DP_ds_read- oder DP_ds_write-Request in Bearbeitung. Ein erneuter Aufruf von DP_ds_read ist erst dann zulässig, wenn das Ergebnis des vorangegangenen Aufrufs mit DP_get_result abgeholt ist. |
| andere | Fehlerhafter Abschluss der Funktion |

### 4.1.14 DP_ds_write

#### Zweck

Mit dieser Funktion kann der Master Daten an einen Slave senden. Das Senden erfolgt parallel zum zyklischen Betrieb.

Es handelt sich hierbei nicht um den Slave-Output, sondern um ein zusätzliches Datenpaket mit einer Adressierung über Slot und Index eines Slave. Der Aufbau und die Bedeutung der Daten kann je nach Slave-Typ variieren.

---

**Hinweis**

Der Schreibvorgang wird **asynchron** ausgeführt. Die Quittung muss mit DP_get_result abgeholt werden.

---

#### Syntax

```
DPR_DWORD DP_ds_write  (DPR_DWORD  user_handle,      // in
                        DPC1_REQ_T       *request    // in
                        DP_ERROR_T       *error );   // out
```

## Parameter

| Name | Beschreibung |
|---|---|
| user_handle | User-Handle, das beim Aufruf DP_open vergeben wurde. |
| request | Zeiger auf DP-V1-Struktur DPC1_REQ_S mit den Einträgen für Datensatz-Schreiben-Auftrag.<br>`typedef struct DPC1_REQ_S`<br>`{`<br>`    DPR_WORD          order_id;    // in`<br>`    DPR_DWORD         c_ref;       // in`<br>`    union`<br>`    {`<br>`      DP_DS_READ_T    dp_ds_read;`<br>`      DP_DS_WRITE_T   dp_ds_write;`<br>`      DP_ALARM_ACK_T  dp_alarm_ack;`<br>`      DP_ENABLE_EVT_T dp_enable_evt;`<br>`      DP_GET_CFG_T    dp_get_cfg;`<br>`    }req;`<br>`} DPC1_REQ_T;`<br>mit<br>`typedef struct DP_DS_WRITE_S`<br>`{`<br>`    DPR_BYTE          slot_number; // in`<br>`    DPR_BYTE          index;       // in`<br>`    DPR_BYTE          length_m;    // in`<br>`    DPR_BYTE data_m[DPR_DPC1_MAX_LENGTH];`<br>`                                   // in`<br>`} DP_DS_WRITE_T;`<br>Order_id ist eine vom Anwenderprogramm frei vergebbare Kennung für den Auftrag. Diese Kennung wird in der asynchronen Quittung unverändert zurückgegeben und kann dafür benützt werden, die Quittung dem Auftrag zuzuordnen.<br>Das Element c_ref spezifiziert den Slave. Über die Hilfsfunktion DP_get_cref(user_id, slv_add) kann der Wert für c_ref ermittelt werden.<br>Zu den Elementen slot_number und index schlagen Sie bitte die jeweilige Slave-Beschreibung nach.<br>Das Element length_m gibt die Länge der an den Slave zu sendenden Daten in data_m an. |
| error | Adresse einer vom Anwenderprogramm bereitgestellten Struktur vom Typ DP_ERROR_T. Die Struktur enthält im Fehlerfall Details zur Fehlerursache (siehe Kapitel 4.4). |

## Return-Wert

| Name | Beschreibung |
|---|---|
| DP_OK_ASYNC | Die Funktionsbearbeitung wurde erfolgreich aktiviert. |
| DP_ERROR_EVENT_NET und Error->error_code == DP_RET_REQ_ACTIV | Es ist noch ein DP_ds_write- oder DP_ds_read-Request in Bearbeitung. Ein erneuter Aufruf von DP_ds_write ist erst dann zulässig, wenn das Ergebnis des vorangegangenen Aufrufs mit DP_get_result abgeholt ist. |
| andere | Fehlerhafter Abschluss der Funktion |

### 4.1.15 DP_fetch_alarm

#### Zweck

DP-Slaves können wichtige Ereignisse in Form von Diagnosedaten schicken. Die Diagnosedaten DPC1-fähiger Slaves können außerdem zusätzlich noch Alarme oder Statusmeldungen enthalten. Die Diagnosedaten werden beim DP-Master zwischengespeichert und können durch diese Funktion ausgelesen werden. Die Funktion kann sowohl für alarmfähige DPC1-Slaves als auch für Standart-DP-Slaves verwendet werden. Es werden, sofern vorhanden, immer die gesamten Diagnosedaten zurückgegeben.

---

**Hinweis 1**

Alle Diagnosetelegramme mit Alarmen werden vom CP 5613/CP 5614 zwischengespeichert und gehen also nicht verloren. Es wird aber jeweils nur das letzte Diagnosetelegramm mit Statusmeldung und das letzte sonstige Diagnosetelegramm ohne Alarm- oder Statusmeldung eines jeden Slave gehalten (sofern vorhanden).

---

**Hinweis 2**

Abhängig von Slave-Typ und Projektierung kann ein DP-Slave bis zu 32 Diagnosetelegramme mit Alarmdaten an den CP 5613/CP 5614 senden.

Das Anwenderprogramm muss jeden Alarm durch einen zugehörigen DP_alarm_ack – Aufruf quittieren (Statusmeldungen aber nicht). Der Aufbau und die Bedeutung der Alarme oder Statusmeldung kann je nach Slave-Typ variieren. Bitte beachten Sie hierzu die Dokumentation des Slave.

---

**Hinweis 3**

Die Funktion meldet die gespeicherten Ereignisse in folgender Priorität zurück:

1. zwischengespeicherte Diagnosedaten mit Alarmen (sofern vorhanden)
2. letzte Diagnosedaten mit/ohne Statusmeldung (sofern vorhanden).

Nach dem Auslesen werden die gespeicherten Daten gelöscht.

Wenn der Slave aus der Bearbeitungsphase herausfällt, werden alle zwischengespeicherte Alarmdaten oder Statusdaten verworfen. In diesem Fall dürfen die entsprechenden Alarme nicht mehr durch DP_alarm_ack–Aufrufe quittiert werden.

---

**Hinweis 4**

Wird im Rückgabeparameter „msg“ der Wert „DP_MSG_TYPE_FAIL“ eingetragen, liegt ein schwerwiegender Fehler beim Alarm-Handling vor. Das Anwenderprogramm muss in diesem Fall einen Restart des Slaves durchführen, wenn zusätzlich das Strukturelement error_decode der Struktur DP_ERROR_T den Wert 0xFF hat (siehe Kapitel 4.1.10).

**Hinweis 5**

Das Anwenderprogramm kann sich über die Funktion DP_enable_event informieren lassen, ob neue Diagnosedaten bzw. Alarme/Statusmeldungen vorhanden sind, die durch DP_fetch_alarm abgeholt werden können. Der Aufruf von DP_enable_event hat den Vorteil, dass nur Diagnosemeldungen in der Betriebsphase des Slave gemeldet werden. Standart-Diagnosemeldungen in der Initialisierungsphase werden – außer bei bestimmten Fehlerfällen – aus Effizienzgründen nicht explizit gemeldet (siehe auch Beschreibung DP_enable_event).

## Syntax

```
DPR_DWORD DP_fetch_alarm (
                 DPR_DWORD         user_handle,     // in
                 DPR_WORD          slv_add,         // in
                 DP_ALARM_EXT_T    *alarm,          // inout
                 DP_ERROR_T        *error);         // out
```

## Parameter

| Name | Beschreibung |
|---|---|
| user_handle | User-Handle, das beim Aufruf DP_open vergeben wurde. |
| slv_add | Adresse des Slave (0-126) |
| Alarm | Zeiger auf die Struktur DP_ALARM_EXT_S mit den ermittelten Diagnosedaten (inklusive Alarmdaten oder Statusdaten).<br><br>`typedef struct DP_ALARM_EXT_S`<br>`{`<br>`  DPR_WORD  msg_filter;    // in`<br>`  DPR_WORD  msg;           // out`<br>`  DPR_WORD  next_msg;      // out`<br>`  DPR_WORD  diag_count     // out`<br>`  DPR_WORD  reserved;      // inout`<br>`  DPR_BYTE  length_s;      // out`<br>`  DPR_BYTE  data_s[DPR_SLAVE_DATA_SIZE];`<br>`                           // out`<br>`} DP_ALARM_EXT_T;`<br><br>**Der Parameter „msg_filter“** gibt die Art der Daten an, die ausgelesen werden. In der vorliegenden Version muss die Kennung DP_MSG_FILTER_ALL eingetragen werden.<br>**Der Parameter „msg“** enthält den Typ der ermittelten Daten:<br>• DP_MSG_NONE Kein Eintrag vorhanden<br>• DP_MSG_ALARM_DIAG Diagnosedaten mit Alarm<br>• DP_MSG_STATUS_DIAG Diagnosedaten mit Statusdaten<br>• DP_MSG_DIAG sonstige Diagnosedaten<br>• DP_MSG_FAIL Fehler beim Alarm-Handling; der Parameter error->errorcode enthält die detaillierte Fehlerursache (siehe unten DP_RET_CP_ ALARM_STATE_xx)<br>**Der Parameter „next_msg“** gibt an, ob noch weitere Diagnosemeldungen gespeichert sind.<br>• next_msg = 0: Es sind keine weiteren Diagnosemeldungen gespeichert.<br>• next_msg = 1: Es sind weitere Diagnosemeldungen gespeichert, die mit DP_fetch_alarm-Aufrufen ausgelesen werden müssen. |

| | |
|---|---|
| | **Der Parameter „diag_count“** gibt den Zählerstand des Diagnose-Counters im Dual Port RAM zum Zeitpunkt des entsprechenden Ereignisses an. Der Wert ist nur relevant, wenn die Funktion Diagnosedaten zurück gibt.<br>**Der Parameter „reserved“** ist für künftige Erweiterungen vorgesehen. Er muss mit dem Wert 0 vorbelegt werden.<br>**Der Parameter „length_s“** enthält die Anzahl der im Array „data_s“ abgelegten Diagnosedaten. |
| error | Adresse einer vom Anwenderprogramm bereitgestellten Struktur vom Typ DP_ERROR_T. Die Struktur enthält im Fehlerfall Details zur Fehlerursache (siehe Kapitel 4.4). |

## Return-Wert

| Name | Beschreibung |
|---|---|
| DP_OK | Erfolgreicher Abschluss der Funktion; der Eintrag in DP_ALARM_EXT_T ist gültig. |
| DP_ERROR_EVENT_NET und error->error_code == DP_RET_CP_ADR_NOT_IN_DB | Fehler: der angegebene Slave existiert nicht in der Datenbasis; der Eintrag in DP_ALARM_EXT_T ist ungültig. |
| DP_ERROR_EVENT_NET und error->error_code == DP_RET_CP_ ALARM_STATE_OVERFLOW | Fehler: Die maximale Anzahl der konfigurierten Alarme wurde vom Slave überschritten. |
| DP_ERROR_EVENT_NET und error->error_code == DP_RET_CP_ ALARM_STATE_INCONSISTENT | Fehler: Der Slave hat mehr als ein Alarm des gleichen Typs gesendet |
| DP_ERROR_EVENT_NET und error->error_code == DP_RET_CP_ ALARM_STATE_WRONG_TYPE | Der Slave hat einen Alarm gesendet, dessen Alarmtyp nicht konfiguriert ist. |
| DP_ERROR_EVENT_NET und error->error_code == DP_RET_CP_ ALARM_STATE_PDU_LENGTH | Der Slave hat einen Alarm oder eine Statusmeldung gesendet, deren Länge größer ist als spezifiziert worden ist. |
| DP_ERROR_EVENT_NET und error->error_code == DP_RET_CP_ ALARM_STATE_PDU_FORMAT | Der Slave hat Diagnosedaten mit Formatfehlern gesendet, so dass eine korrekte Dekodierung des Alarms oder des Status nicht möglich ist. |
| andere | Fehlerhafter Abschluss der Funktion; der Eintrag in DP_ALARM_EXT_T ist ungültig. |

### 4.1.16 DP_alarm_ack

#### Zweck

Bei bestimmten Situationen können DP-Slaves mit DPC1-Funktionalität Alarme melden. Diese Alarme werden vom DP-Master als Teil der Diagnosedaten empfangen und zwischengespeichert. Alarme müssen dem Slave vom Master-DP-Anwenderprogramm explizit durch die Funktion DP_alarm_ack quittiert werden, jeder für sich.

---

**Hinweis**

Der Sendevorgang wird **asynchron** ausgeführt. Die Quittung muss mit DP_get_result abgeholt werden.

---

#### Syntax

```
DPR_DWORD  DP_alarm_ack(DPR_DWORD  user_handle,      // in
                        DPC1_REQ_T *request,         // in
                        DP_ERROR_T *error );         // out
```

## Parameter

<table>
<tr><th>Name</th><th>Beschreibung</th></tr>
<tr><td>user_handle</td><td>User-Handle, das beim Aufruf DP_open vergeben wurde.</td></tr>
<tr><td>request</td><td>Zeiger auf DP-V1-Struktur DPC1_REQ_S mit den Einträgen für Alarm-Acknowledge-Auftrag.
<pre>
typedef struct DPC1_REQ_S
{
     DPR_WORD         order_id;   // in
     DPR_DWORD        c_ref;      // in
     union
     {
     DP_DS_READ_T     dp_ds_read;
     DP_DS_WRITE_T    dp_ds_write;
     DP_ALARM_ACK_T   dp_alarm_ack;
     DP_ENABLE_EVT_T  dp_enable_evt;
     DP_GET_CFG_T     dp_get_cfg;
     }req;
} DPC1_REQ_T;
</pre>
mit
<pre>
typedef struct DP_ALARM_ACK_S
{
  DPR_BYTE slot_number;           // in
  DPR_BYTE alarm_type;            // in
  DPR_BYTE  specifier;            // in
} DP_ALARM_ACK_T;
</pre>
Order_id ist eine vom Anwenderprogramm frei vergebbare Kennung für den Auftrag. Diese Kennung wird in der asynchronen Quittung unverändert zurückgegeben und kann dafür benützt werden, die Quittung dem Auftrag zuzuordnen.<br><br>
Das Element c_ref spezifiziert den Slave. Über die Hilfsfunktion DP_get_cref(user_id, slv_add) kann der Wert für c_ref ermittelt werden.<br><br>
Die Elemente slot_number, alarm_type und specifier stammen aus dem empfangenen Slave-Alarm.</td></tr>
<tr><td>error</td><td>Adresse einer vom Anwenderprogramm bereitgestellten Struktur vom Typ DP_ERROR_T. Die Struktur enthält im Fehlerfall Details zur Fehlerursache (siehe Kapitel 4.4).</td></tr>
</table>

## Return-Wert

| Name | Beschreibung |
|---|---|
| DP_OK_ASYNC | Die Funktionsbearbeitung wurde erfolgreich aktiviert. |
| DP_ERROR_EVENT_NET und error->error_code == DP_RET_REQ_ACTIV | Es ist noch ein DP_alarm_ack-Request in Bearbeitung. Ein erneuter Aufruf von DP_alarm_ack ist erst dann zulässig, wenn das Ergebnis des vorangegangenen Aufrufs mit DP_get_result abgeholt ist. |
| andere | Fehlerhafter Abschluss der Funktion |

### 4.1.17 DP_get_actual_cfg

#### Zweck

Mit dieser Funktion können die aktuellen Konfigurationsdaten von einem Slave ausgelesen werden. Die Daten werden über ein spezielles DP-Telegramm vom Slave angefordert. Dadurch kann erkannt werden, ob die projektierte Konfiguration (Datenbasis) mit der tatsächlichen Konfiguration übereinstimmt.

Diese Funktion kann bei modularen Slaves verwendet werden, um den tatsächlichen Ausbau zu bestimmen.

---

**Hinweis**

Der Lesevorgang wird **asynchron** ausgeführt. Die Quittung muss mit DP_get_result abgeholt werden.

---

#### Syntax

```
DPR_DWORD  DP_get_actual_cfg (DPR_DWORD  user_handle, // in
                              DPC1_REQ_T  *request,   // in
                              DP_ERROR_T  *error);    // out
```

## Parameter

| Name | Beschreibung |
|---|---|
| user_handle | User-Handle, das beim Aufruf DP_open vergeben wurde. |
| request | Zeiger auf DP-V1-Struktur DPC1_REQ_S mit den Einträ-gen für den DP_get_actual_cfg-Auftrag.<br><br>typedef struct DPC1_REQ_S<br>{<br>DPR_WORD order_id; // in<br>DPR_DWORD c_ref; // in<br>union<br>{<br>DP_DS_READ_T dp_ds_read;<br>DP_DS_WRITE_T dp_ds_write;<br>DP_ALARM_ACK_T dp_alarm_ack;<br>DP_ENABLE_EVT_T dp_enable_evt;<br>DP_GET_CFG_T dp_get_cfg;<br>}req;<br>} DPC1_REQ_T;<br><br>mit<br><br>typedef struct DP_GET_CFG_S<br>{<br>DPR_BYTE length_s; // out<br>DPR_BYTE data_s[DPR_SLAVE_DATA_SIZE];<br>// out<br>} DP_GET_CFG_T;<br><br>Order_id ist eine vom Anwenderprogramm frei vergebbare Kennung für den Auftrag. Diese Kennung wird in der asyn-chronen Quittung unverändert zurückgegeben und kann dafür benützt werden, die Quittung dem Auftrag zuzuord-nen.<br><br>Das Element c_ref spezifiziert den Slave. Über die Hilfsfunktion DP_get_cref(user_id, slv_add) kann der Wert für c_ref ermittelt werden.<br><br>Das Element length_s gibt die Länge der vom Slave emp-fangenen Daten an. In data_s werden die empfangenen Daten eingetragen (siehe Kap. 4.7). |
| error | Adresse einer vom Anwenderprogramm bereitgestellten Struktur vom Typ DP_ERROR_T. Die Struktur enthält im Fehlerfall Details zur Fehlerursache (siehe Kapitel 4.4). |

## Return-Wert

| Name | Beschreibung |
|---|---|
| DP_OK_ASYNC | Die Funktionsbearbeitung wurde erfolgreich aktiviert. |
| DP_ERROR_EVENT_NET und error->error_code == DP_RET_REQ_ACTIV | Es ist noch ein DP_get_actual_cfg-Request in Bearbeitung. Ein erneuter Aufruf von DP_get_actual_cfg ist erst dann zulässig, wenn das Ergebnis des vorangegangenen Aufrufs mit DP_get_result abgeholt ist. |
| andere | Fehlerhafter Abschluss der Funktion |

### 4.1.18 DP_enable_event

#### Zweck

Der Aufruf dieser Funktion bewirkt, dass folgende wichtige Ereignisse (Software-Events) im DP-Master der DP-Anwendung explizit gemeldet werden können:

- Diagnosedaten, Alarme und Statusmeldungen können abgeholt werden
- Slaves sind in die zyklische DP-Bearbeitung aufgenommen worden (Betriebsbereitschaft) bzw. sind ausgefallen
- Der Zustand des Masters hat sich geändert

Die Funktion wartet nicht auf die Ereignisse, sondern meldet Ihr Programm nur dafür an. Zum Abholen der Ereignisse benutzen Sie bitte DP_get_result.

---

**Hinweis 1**

Mit dem Aufruf dieser Funktion erklärt sich Ihr Anwenderprogramm lediglich empfangsbereit für den angegebenen Event-Typ. Die eigentliche Ereignismeldung muss durch DP_get_result abgeholt werden.

---

**Hinweis 2**

Nach Empfang einer DP_enable_event–Confirmation (siehe DP_get_result) muss die Funktion DP_enable_event erneut aufgerufen werden, wenn das Anwenderprogramm sich über neue Events informieren lassen will. Die Ereignisse DP_DIAG_ALARM und DP_SLV_STATE werden zwischen Empfang der Confirmation und erneutem Aufruf von DP_enable_event zwischengespeichert und gehen somit nicht verloren.

---

**Hinweis 3**

Beim erstmaligen Aufruf von DP_enable_event mit aktiviertem Selektor DP_SLV_STATE sind folgende Punkte zu beachten:

- Sind zum Zeitpunkt des Aufrufs Slaves betriebsbereit (READY-Zustand), so wird für diese Slaves sofort die Ereignismeldung DP_SLAVE_ENTER ausgelöst.
- Sind zum Zeitpunkt des Aufrufs keine Slaves betriebsbereit (NOT READY-Zustand), so wird keine Ereignismeldung DP_SLAVE_EXIT ausgelöst.

---

---

**Hinweis 4**

Wird für einen Slave in aufeinanderfolgenden DP_enable_event-Confirmations das Ereignis DP_SLAVE_ENTER bzw. DP_SLAVE_EXIT gemeldet (ohne zwischenzeitliches DP_SLAVE_EXIT bzw. DP_SLAVE_ENTER) bedeutet dies folgendes:

- Mehrfaches DP_SLAVE_ENTER: Der Slave wechselte zwischenzeitlich in den NOT READY-Zustand und danach wieder zurück in den READY-Zustand.
- Mehrfaches DP_SLAVE_EXIT: Der Slave wechselte zwischenzeitlich in den READY-Zustand und danach wieder zurück in den NOT READY-Zustand.

---

**Hinweis 5**

Ist der Slave im Zustand NOT_READY (d.h. in der Initialisierungsphase) und der Selektor DP_DIAG_ALARM aktiviert, dann wird die Diagnose nur dann gemeldet, wenn der Slave in seinem Diagnosetelegramm Diag.Prm_Fault, Diag.Cfg_Fault oder Diag.Ext_Diag meldet (zu den Formaten siehe Kap. 4.6).

---

**Hinweis 6**

Ist der DP-Master zum Zeitpunkt des Aufrufs von DP_enable_event mit Selector DP_MST_STATE nicht im zu überwachenden Zustand mst_state, dann wird der Event sofort ausgelöst.

---

**Hinweis 7**

Der Event DP_ALARM_STATUS wird in aufeinanderfolgenden DP_enable_event-Confirmations solange gemeldet, bis alle Alarm- oder Statusdaten mit DP_fetch_alarm ausgelesen worden sind.

---

## Syntax

```
DPR_DWORD  DP_enable_event   (DPR_DWORD  user_handle,// in
                              DPC1_REQ_T  *request,   // in
                              DP_ERROR_T  *error );   // out
```

## Parameter

<table>
<tr><th>Name</th><th>Beschreibung</th></tr>
<tr><td>user_handle</td><td>User-Handle, das beim Aufruf DP_open vergeben wurde.</td></tr>
<tr><td>Request</td><td>Zeiger auf die DP-V1-Struktur DPC1_REQ_S mit den Einträgen für den Event-Mechanismus.<br><pre>typedef struct DPC1_REQ_S
{
     DPR_WORD    order_id;   // in
     DPR_DWORD   c_ref;      // reserved
     union
     {
          DP_DS_READ_T      dp_ds_read;
          DP_DS_WRITE_T     dp_ds_write;
          DP_ALARM_ACK_T    dp_alarm_ack;
          DP_ENABLE_EVT_T   dp_enable_evt;
          DP_GET_CFG_T      dp_get_cfg;
     }req;
} DPC1_REQ_T;</pre>Order_id ist eine vom Anwenderprogramm frei vergebbare Kennung für den Auftrag. Diese Kennung wird in der asynchronen Quittung unverändert zurückgegeben und kann dafür benützt werden, die Quittung dem Auftrag zuzuordnen.<br>Das Element c_ref ist hier reserviert.<br>In der Union „req“ ist nur die Variante „dp_enable_event“ relevant, die weiter unten beschrieben wird.</td></tr>
<tr><td>error</td><td>Adresse einer vom Anwenderprogramm bereitgestellten Struktur vom Typ DP_ERROR_T. Die Struktur enthält im Fehlerfall Details zur Fehlerursache (siehe Kapitel 4.4).</td></tr>
</table>

## Aufbau des Strukturelements request->req.dp_enable_evt

```
typedef struct   DP_ENABLE_EVT_S
{
     DPR_DWORD   selector;                      // in
     DPR_BYTE    mst_state;                     // in
     DPR_BYTE    event [DPR_MAX_SLAVE_ADDR];    // out
     DPR_BYTE    mst_event;                     // out
     DPR_BYTE    new_mst_state;                 // out
} DP_ENABLE_EVT_T;
```

## Beschreibung der Elemente von request->req.dp_enable_evt

| Element | Beschreibung |
|---|---|
| selector | Das Element selector wählt den Typ der zu meldenden Ereignisse aus. Die Werte können einzeln oder durch „Oder-Verknüpfung“ angegeben werden:<br>• DP_DIAG_ALARM – Benachrichtigung bei Diagnose oder Alarmen<br>• DP_SLAVE_STATE – Benachrichtigung, wenn der Slave in die Bearbeitung aufgenommen wird oder aus ihr herausfällt.<br>• DP_MST_STATE – Benachrichtigung, wenn der Master einen anderen Status hat oder ein nimmt als im Vorgabeparameter mst_state eingetragen ist. |
| mst_state | Das Element mst_state ist nur relevant, wenn in selector die Kennung DP_MST_STATE eingetragen ist. Es muss derjenige Master-Betriebszustand eingetragen werden, bei dessen Änderung die Funktion DP_enable_event das Ereignis DP_MST_STATE_CHG melden soll. Es kann einer der folgenden Werte gewählt werden:<br>• DP_OFFLINE<br>• DP_STOP<br>• DP_CLEAR<br>• DP_AUTOCLEAR<br>• DP_OPERATE |

Fortsetzung der Tabelle auf der nächsten Seite

Fortsetzung der Tabelle von der letzten Seite

| Element | Beschreibung |
|---|---|
| event | Das Array event enthält nach Durchführung des Auftrags pro Slave-Adresse ein Element mit den Ereignissen, die diesen Slave betreffen. So enthält `event[5]` z. B. Informationen über den Slave 5.<br>Folgende Ereigniskennungen können pro Slave-Adresse zurückgegeben werden, ggf. auch mehrere mit „Oder“ verknüpft:<br>• 0 — kein Ereignis<br>• DP_DIAG — Diagnose ist eingetroffen und kann durch DP_fetch_alarm ausgelesen werden.<br>• DP_ALARM_STATUS — Diagnose mit Alarm- oder Statusdaten wurden zwischengespeichert und müssen mit DP_fetch_alarm ausgelesen werden.<br>• DP_SLAVE_ENTER — Der Slave wurde in die Bearbeitung aufgenommen.<br>• DP_SLAVE_EXIT — Der Slave ist aus der Bearbeitung herausgefallen. |
| mst_event | Das Element mst_event enthält die Kennung DP_MST_STATE_CHG, wenn in selector der Wert DP_MST_STATE eingetragen ist und der Master-Zustand sich gegenüber dem in mst_state vorgegebenen Wert geändert hat; ansonsten den Wert 0. |
| new_mst_state | new_mst_state enthält den aktuellen Master-Zustand. |

## Return-Wert

| Name | Beschreibung |
|---|---|
| DP_OK_ASYNC | Die Funktionsbearbeitung wurde erfolgreich aktiviert. |
| DP_ERROR_EVENT_NET und error->error_code == DP_RET_REQ_ACTIV | Es ist noch ein DP_enable_event-Request in Bearbeitung. Ein erneuter Aufruf von DP_enable_event ist erst dann zulässig, wenn<br>• Eine DP_enable_event-Confirmation empfangen wurde<br>• Der Aufruf durch einen Aufruf von DP_disable_event rückgängig gemacht wurde |
| andere | Fehlerhafter Abschluss der Funktion |

### 4.1.19 DP_disable_event

#### Zweck

Durch diese Funktion kann ein vorangegangener Aufruf von DP_enable_event rückgängig gemacht werden. Damit kann die zuvor spezifizierte Empfangsbereitschaft beendet werden, ohne dass ein entsprechendes Ereignis eingetreten ist. Die Quittung des beendeten Aufrufs muss mit DP_get_result abgeholt werden. Anschließend kann DP_enable_event, z. B. mit geänderten Bedingungen, noch einmal aufgerufen werden.

#### Syntax

```
DPR_DWORD  DP_disable_event  (DPR_DWORD  user_handle,// in
                              DP_ERROR_T *error );   // out
```

#### Parameter

| Name | Beschreibung |
|---|---|
| user_handle | User-Handle, das beim Aufruf DP_open vergeben wurde. |
| error | Adresse einer vom Anwenderprogramm bereitgestellten Struktur vom Typ DP_ERROR_T. Die Struktur enthält im Fehlerfall Details zur Fehlerursache (siehe Kapitel 4.4). |

#### Return-Wert

| Name | Beschreibung |
|---|---|
| DP_OK | Die Funktionsbearbeitung wurde erfolgreich abgesetzt. |
| DP_ERROR_EVENT_NET und error->error_code == DP_RET_CP_NO_EVT_PENDING | Es ist kein DP_enable_event - Request in Bearbeitung |
| andere | Fehlerhafter Abschluss der Funktion |

### 4.1.20 DP_get_result

#### Zweck

Diese Funktion holt die Quittung eines asynchronen Auftrags ab. Für jeden asynchronen Auftrag muss die zugehörige Quittung durch diese Funktion abgeholt werden.

Abhängig von der übergebenen Timeout-Zeit kehrt die Funktion **sofort** oder nach Ablauf der Überwachungszeit zurück. Wird als Timeout-Wert „unendlich“ eingetragen, wartet DP_get_result solange, bis eine Quittung eingetroffen ist.

---

**Hinweis 1**

Wird beim Absetzen von DP_close noch auf asynchrone Quittungen gewartet, wird DP_get_result mit der Fehlerklasse DP_ERROR_USR_ABORT beendet.

---

**Hinweis 2**

Die Quittungen können prinzipiell in einer beliebigen Reihenfolge eintreffen. Hierzu eine Beispielsequenz:

- Das Anwenderprogramm setzt einen DP_enable_event-Aufruf ab, um beim Eintreffen von Alarmen unterrichtet zu werden.
- Das Anwenderprogramm setzt einen DP_ds_write-Aufruf ab, um z. B. Slave 10 nachzuparametrieren.
- Daraufhin ruft das Anwenderprogramm in einem zweiten Thread die Funktion DP_get_result auf, und erhält die Quittung für den DP_ds_write-Auftrag.
- Erst beim nächsten Aufruf von DP_get_result wird mit der DP_enable_event-Quittung gemeldet, dass ein Alarm für den Slave 10 vorliegt.

---

**Hinweis 3**

Wird bei der Fehlerklasse DP_ERROR_EVENT_NET das Strukturelement error_docode der Struktur DP_ERROR_T auf den Wert 0xFF gesetzt, so liegt ein Kommunikationsfehler vor.

Bei einem DPC1-Auftrag (req_type = DP_DS_READ, DP_DS_WRITE oder DP_ALARM_ACK) muss ein DPC1-fähiger Slave danach durch den Aufruf DP_slv_state (Restart) wieder neu initialisiert werden, sofern der Master sich im Zustand Operate oder Clear befindet.

---

## Syntax

```
DPR_DWORD  DP_get_result(DPR_DWORD user_handle,      // in
                        DPR_DWORD   timeout,          // in
                        DPR_WORD    *req_type,        // out
                        DPC1_REQ_T  *result,          // out
                        DP_ERROR_T  *error            // out
```

## Parameter

| Name | Beschreibung |
|---|---|
| user_handle | User-Handle, das beim Aufruf DP_open vergeben wurde. |
| timeout | Dauer der maximalen Wartezeit (in Millisekunden), bis die Funktion zurückkehrt.<br>Grenzwerte:<br>0: keine Wartezeit - Funktion kehrt sofort zurück<br>0x7FFFFFFE: maximale Wartezeit<br>DP_TIMEOUT_FOREVER: „unendliche“ Wartezeit |
| req_type | Adresse einer Variablen für den Auftragstyp. Nach Empfang einer Quittung wird hier der Auftragstyp eingetragen:<br>• DP_NO_CNF: keine Quittung empfangen<br>• DP_DS_READ: Quittung für DP_ds_read<br>• DP_DS_WRITE: Quittung für DP_ds_write<br>• DP_ALARM_ACK: Quittung für DP_alarm_ack<br>• DP_ENABLE_EVENT Quittung für DP_enable_event<br>• DP_GET_CFG Quittung für DP_get_actual_cfg |
| result | Der Parameter result zeigt auf eine Struktur vom Typ DPC1_REQ_T - Beim Empfang einer Confirmation werden in die zugehörigen Substrukturen dieser Struktur die Quittungsparameter eingetragen (entsprechend dem Eintrag in req_type). |
| error | Adresse einer vom Anwenderprogramm bereitgestellten Struktur vom Typ DP_ERROR_T. Die Struktur enthält im Fehlerfall Details zur Fehlerursache (siehe Kapitel 4.4). |

## Return-Wert

| Name | Beschreibung |
|---|---|
| DP_OK | Es wurde eine Confirmation empfangen - Die Werte in req_type und result identifizieren den Auftrag. Der zugehörige Auftrag wurde ohne Fehler beendet. |
| DP_ERROR_EVENT_NET und error->error_code == DP_RET_TIMEOUT | Keine Quittung verfügbar. |
| DP_ ERROR_EVENT_NET und error->error_code == DP_RET_CP_REQ_NEG | Kommunikationsfehler (Slave antwortet nicht oder der Zugangspunkt ist beim Slave nicht bereit); siehe Hinweis 3 (oben). |
| DP_ ERROR_EVENT_NET und error->error_code == DP_RET_CP_TIME_OUT | Kommunikationsfehler (Slave antwortet nicht innerhalb der Timeout-Zeit.); siehe Hinweis 3 (oben). |
| DP_ ERROR_EVENT_NET und error->error_code == DP_RET_CP_REQ_RE oder DP_RET_CP_MM_xx (xx = AD, DI, EA, FE, IP, LE, NC, NE, NI, RE, SC, SE) | Kommunikationsfehler (lokaler oder remoter DP-Fehler); siehe Hinweis 3 (oben). |
| DP_ ERROR_EVENT_NET und error->error_code == DP_RET_CP_SLV_NOT_ IN_DATA | Der Auftrag kann nicht gesendet werden, da der Slave nicht bereit ist. |
| DP_ERROR_EVENT | Es wurde eine Rückmeldung von einem DP-V1-Slave empfangen. Die Auswertung solcher Rückmeldungen ist in Kapitel 4.4.1 beschrieben. |
| DP_ERROR_EVENT_NET, sonstige oder DP_ERROR_RES | Es wurde eine Confirmation empfangen. Der zugehörige Auftrag wurde mit einem Fehler beendet. Die Rückgabestruktur error enthält Details zur Fehlerursache. |
| DP_ERROR_CI, DP_ERROR_USR_ABORT | Der Funktionsaufruf wurde mit einem Fehler beendet. Es wurde keine Confirmation empfangen. Der Parameter result ist nicht gültig. Die Rückgabestruktur error enthält Details zur Fehlerursache. |

### 4.1.21 DP_get_cref

#### Zweck

Diese Funktion ermittelt aus der Slave-Adresse und dem user_handle die c_ref für die DPC1-Funktionen. Die c_ref muss dort in der Struktur DPC1_REQ_S als Vorgabeparameter eingetragen werden.

---

**Hinweis**

Die c_ref für einen Slave muss nur einmal ermittelt werden. Es ist nicht erforderlich, vor jeder DPC1-Funktion DP_get_cref erneut aufzurufen.

---

#### Syntax

```
DPR_DWORD  DP_get_cref (DPR_DWORD  user_handle,      // in
                        DPR_WORD   slv_add,          // in
                        DPR_DWORD  *c_ref,           // out
                        DP_ERROR_T *error );         // out
```

#### Parameter

| Name | Beschreibung |
|---|---|
| user_handle | User-Handle, das beim Aufruf DP_open vergeben wurde. |
| c_ref | Nach erfolgreichem Abschluss der Funktion enthält *c_ref die angeforderte Referenz. |
| slv_add | Slave-Adresse |
| error | Adresse einer vom Anwenderprogramm bereitgestellten Struktur vom Typ DP_ERROR_T - Die Struktur enthält im Fehlerfall Details zur Fehlerursache (siehe Kapitel 4.4). |

#### Return-Wert

| Name | Beschreibung |
|---|---|
| DP_OK | Die c_ref konnte ermittelt werden. |
| andere | Fehlerhafter Abschluss der Funktion, keine Quittung verfügbar |

## 4.1.22 DP_init_sema_object

### Zweck

Diese Funktion richtet ein Semaphor für ein Ereignis vom CP 5613/CP 5614 ein. An Win32-API-Funktionen (WaitForMultipleObjects, WaitForSingleOblect, MsgWaitForMultipleObjects) kann ein Anwenderprogramm an diesen Semaphoren warten, bis ein Ereignis eingetroffen ist.

Semaphore sind Synchronisationsobjekte, die auch die Win32-API-Schnittstelle unterstützt. Mit ihnen kann man in Threads oder in Prozessen auf das Eintreffen von Ereignissen warten.

---

**Hinweis 1**

Um für den CP 5614 Semaphore für Master- und Slave-Betrieb gleichzeitig zu verwenden, erzeugen Sie ein Master-Semaphor vom Typ DP_OBJECT_TYPE_ASYNC mit dem User-Handle des DP_open-Aufrufs und dann noch eines vom Typ DP_OBJECT_TYPE_ASYNC, aber mit dem User-Handle des DPS_open-Aufrufs.

---

**Hinweis 2**

Ein mit DP_init_sema_object angelegtes Semaphor darf nur mit der Funktion DP_delete_sema_objet gelöscht werden. Verwenden Sie keinesfalls Win32-API-Funktionen.

---

### Syntax

```
DPR_DWORD  DP_init_sema_object(
                     DPR_DWORD   user_handle,     // in
                     DPR_DWORD   sema_type,       // in
                     DPR_DWORD   *sema_handle,    // out
                     DP_ERROR_T  *error);         // out
```

## Parameter

| Name | Beschreibung |
|---|---|
| user_handle | User-Handle, das beim Aufruf DP_open vergeben wurde. |
| sema_type | Art des Semaphors:<br>• DP_OBJECT_TYPE_INPUT_CHANGE - Hardware-Event bei Änderung von Slave-Eingabedaten<br>• DP_OBJECT_TYPE_DIAG_CHANGE - Hardware-Event bei Diagnosedaten<br>• DP_OBJECT_TYPE_CYCLE_INT - Hardware-Event beim Start eines neuen DP-Zyklus<br>• DP_OBJECT_TYPE_CYCLE_END - Hardware-Event beim Ende eines neuen DP-Zyklus<br>• DP_OBJECT_TYPE_ASYNC - Software-Event bei Abschluss eines asynchronen Aufrufs, kann mit DP_get_result abgeholt werden<br>• DP_OBJECT_TYPE_FAST_LOGIC - Hardware-Event für Fast Logic<br>Bis auf DP_OBJECT_TYPE_ASYNC können die Semaphore nur je einmal pro CP erzeugt werden. |
| sema_handle | Adresse einer Variablen für das Semaphorobjekt - Das zurückgegebene Semaphorobjekt muss bei den Win32-API-Funktionen verwendet werden. |
| error | Adresse einer vom Anwenderprogramm bereitgestellten Struktur vom Typ DP_ERROR_T - Die Struktur enthält im Fehlerfall Details zur Fehlerursache (siehe Kapitel 4.4). |

## Return-Wert

| Name | Beschreibung |
|---|---|
| DP_OK | Das Semaphor konnte eingerichtet werden. |
| DP_ ERROR_CI und error->error_code == CI_RET_SEMA_TWICE | Es existiert bereits ein Semaphorobjekt für diesen User-Handle. |
| andere | Fehlerhafter Abschluss der Funktion |

### 4.1.23 DP_delete_sema_object

#### Zweck

Diese Funktion entfernt ein zuvor eingerichtetes Semaphor.

#### Syntax

```
DPR_DWORD DP_delete_sema_object (
                       DPR_DWORD   user_handle,      // in
                       DPR_DWORD sema_handle,        // in
                       DP_ERROR_T  *error);          // out
```

#### Parameter

| Name | Beschreibung |
|---|---|
| user_handle | User-Handle, das beim Aufruf DP_open vergeben wurde. |
| sema_handle | Semaphor-Handle |
| error | Adresse einer vom Anwenderprogramm bereitgestellten Struktur vom Typ DP_ERROR_T - Die Struktur enthält im Fehlerfall Details zur Fehlerursache (siehe Kapitel 4.4). |

#### Return-Wert

| Name | Beschreibung |
|---|---|
| DP_OK | Das Semaphor konnte entfernt werden. |
| andere | Fehlerhafter Abschluss der Funktion. |

## 4.1.24 DP_fast_logic_on

### Zweck

Mit dieser Funktion kann ein DP-Master-Anwenderprogramm die „Fast Logic“-Funktionalität des CP 5613/CP 5614 aktivieren. Durch diese Funktionalität kann ein Eingangsbyte eines Slave X mit einem Ausgabebyte eines Slave Y verknüpft werden. Bei Zutreffen der vorgegebenen Trigger-Bedingung beim Slave X schreibt die CP 5613/CP 5614-Firmware selbsttätig einen vorgegebenen Ausgabewert in das Ausgabebyte eines Slave Y.

Die Reaktionszeit der „Fast Logic“-Funktionalität ist unabhängig von der Geschwindigkeit und Belastung des eingesetzten Rechners. Daher kann sie zur Steuerung von zeitkritischen Aufgaben eingesetzt werden.

Es können bis zu vier unabhängige Fast-Logic-Trigger-Bedingungen pro CP 5613/CP 5614 aktiviert werden.



**Warnung 1**

Die Fast Logic funktioniert nur dann korrekt, wenn der DP-Master im Zustand OPERATE und die beteiligten Slaves im Zustand READY sind. Deshalb sollte ein Fast-Logic-Trigger von dem DP-Anwenderprogramm auch erst aktiviert werden, wenn das Anwenderprogramm den DP-Master in den Zustand OPERATE hochgefahren hat.



**Warnung 2**

Beachten Sie, dass solange Fast-Logic-Trigger aktiv sind, **KEIN** DP-Anwenderprogramm schreibend auf die Ausgangsbytes zugreifen darf, die über die Fast Logic mit Eingangsbytes verknüpft sind.

**Hinweis 1**

Nachdem ein Fast-Logic-Trigger ausgelöst wurde, wird er anschließend automatisch deaktiviert. Ein erneutes Aktivieren des Triggers mit DP_fast_logic_on ist erst dann möglich, wenn dieser zuvor vom DP-Anwenderprogramm im Dualport RAM quittiert wurde (siehe Beispiel in Kapitel 4.3.10 „Fast-Logic-Status-Abfragen“).

Die Fast Logic wird ebenfalls automatisch deaktiviert, wenn das Anwenderprogramm sich mit der Funktion DP_close abmeldet.

**Hinweis 2**

Bei Multiapplikationsbetrieb kann zu einem Zeitpunkt immer nur ein DP-Master-Anwenderprogramm die Fast-Logic-Funktionalität des CP 5613/CP 5614 nutzen.

---

**Hinweis 3**

In dem Kapitel 4.3.10 „Fast-Logic-Status-Abfragen“ wird beschrieben wie das Anwenderprogramm den aktuellen Status der Fast-Logic-Trigger abfragen kann.

Für Fast Logic existiert analog zu anderen Ereignissen ein Semaphor, das bei Eintreten eines Fast-Logic-Triggers auf Freigabe geschaltet wird. Siehe Kapitel 4.1.22, „DP_init_sema_object“.

---

**Hinweis 4**

Nachdem ein Fast-Logic-Trigger vom CP 5613/CP 5614 ausgelöst wurde, sollte das DP-Anwenderprogramm prüfen, ob der Slave, dessen Ausgabebyte durch die Fast Logic geschrieben wurde, noch im Zustand READY ist. Ist das nicht der Fall, übernimmt der Slave das geänderte Ausgangsbyte erst, wenn er wieder in den Zustand READY geht. (siehe Kapitel 4.3.5, „Zustand eines DP-Slave feststellen“)

---

**Hinweis 5**

Die Zeit zwischen der Änderung im Prozessabbild des CP 5613 für das selektierte Eingangsbyte und dem Setzen des selektierten Ausgangsbytes beträgt ca. 40 µs.

---

## Syntax

```
DPR_DWORD DP_fast_logic_on (
                 DPR_DWORD         user_handle,      // in
                 DPR_WORD          fast_logic_id,    // in
                 DP_FAST_LOGIC_T   *fast_logic,      // in
                 DP_ERROR_T        *error);          // out
```

## Parameter

| Name | Beschreibung |
|---|---|
| user_handle | User-Handle, das beim Aufruf DP_open vergeben wurde. |
| fast_logic_id | Auswahl des Fast-Logic-Triggers<br>Wertebereich: 0 bis (DPR_MAX_FASTLOGIC_CNT – 1) - DPR_MAX_FASTLOGIC_CNT ist auf 4 festgelegt. |

Fortsetzung der Tabelle auf der nächsten Seite

Fortsetzung der Tabelle von der letzten Seite

<table>
<tr><td>fast_logic</td><td>Zeiger auf die Struktur DP_FAST_LOGIC_S mit der die Trigger-Bedingung eingestellt werden kann.<br>
<pre>
typedef struct DP_FAST_LOGIC_S
{
        DPR_BYTE    slave_addr_in_byte;    // in
        DPR_BYTE    index_in_byte;         // in
        DPR_BYTE    cmp_value_in_byte;     // in
        DPR_BYTE    mask_in_byte;          // in

        DPR_BYTE    slave_addr_out_byte;   // in
        DPR_BYTE    index_out_byte;        // in
        DPR_BYTE    value_out_byte;        // in
        DPR_BYTE    mask_out_byte;         // in
} DP_FAST_LOGIC_T;
</pre>
slave_addr_in_byte gibt die Adresse des Slaves an, dessen Eingänge für den Trigger selektiert werden.<br>
index_in_byte gibt den Index des Eingangsbytes des Triggers an.<br>
cmp_value_in_byte gibt den Vergleichswert für das Eingangsbyte an.<br>
Mit mask_in_byte können einzelne Bits im Eingangsbyte maskiert werden, so dass sie beim Vergleich nicht berücksichtigt werden. Maskiert wird durch eine 1 im entsprechenden Bit, d. h. für mask_in_byte == 0x00 werden alle Bits von cmp_value_in_byte für den Vergleich herangezogen.<br>
Der Trigger wird ausgelöst, wenn alle nichtmaskierten Bits im selektierten Eingangsbyte gleich den Bits in cmp_value_in_byte sind.<br>
slave_addr_out_byte selektiert den Slave, dessen Ausgangsbyte beim Eintreffen der Triggerbedingung verändert werden soll.<br>
index_out_byte gibt den Index des Ausgangsbytes an.<br>
value_out_byte gibt den Wert an, der in das Ausgangsbyte geschrieben werden soll.<br>
Mit mask_out_byte können einzelne Bits im Ausgangsbyte maskiert werden, so dass sie beim Eintreffen der Triggerbedingung nicht verändert werden. Maskiert wird durch eine 1 im entsprechenden Bit, d. h. für mask_out_byte == 0x00 werden alle Bits von value_out_byte in das selektierte Ausgangsbyte geschrieben.</td></tr>
<tr><td>error</td><td>Adresse einer vom Anwenderprogramm bereitgestellten Struktur vom Typ DP_ERROR_T - Die Struktur enthält im Fehlerfall Details zur Fehlerursache (siehe Kapitel 4.4).</td></tr>
</table>

## Return-Wert

| Name | Beschreibung |
|---|---|
| DP_OK | Der Fast-Logic-Trigger wurde aktiviert. |
| DP_ ERROR_CI und<br>error->error_code ==<br>CI_RET_FL_INV_ID | Parameter fast_logic_id ungültig. |
| DP_ ERROR_CI und<br>error->error_code ==<br>CI_RET_FL_INV_ADDR_IN_BYTE | Ungültige Adresse des Eingabe-Slaves. |
| DP_ ERROR_CI und<br>error->error_code ==<br>CI_RET_FL_INV_ADDR_OUT_BYTE | Ungültige Adresse des Ausgabe-Slaves. |
| DP_ ERROR_CI und<br>error->error_code ==<br>CI_RET_FL_SLAVE_IN_ NOT_IN_DB | Eingabe-Slave nicht in der Datenbasis. |
| DP_ ERROR_CI und<br>error->error_code ==<br>CI_RET_FL_SLAVE_OUT_<br>NOT_IN_DB | Ausgabe-Slave nicht in der Datenbasis. |
| DP_ ERROR_CI und<br>Error->error_code ==<br>CI_RET_FL_INV_INDEX_ IN_BYTE | Ungültiges Eingabebyte. |
| DP_ ERROR_CI und<br>Error->error_code ==<br>CI_RET_FL_INV_INDEX_OUT_BYTE | Ungültiges Ausgabebyte. |
| DP_ ERROR_CI und<br>Error->error_code ==<br>CI_RET_FL_ALREADY_ON | Fast Logic ist bereits aktiviert. |
| DP_ ERROR_CI und<br>Error->error_code ==<br>CI_RET_FL_INV_IN_MASK | Ungültige Maske für das Eingabebyte. |
| DP_ ERROR_CI und<br>error->error_code ==<br>CI_RET_FL_INV_OUT_MASK | Ungültige Maske für das Ausgabebyte. |
| DP_ ERROR_CI und<br>error->error_code ==<br>CI_RET_FL_DOUBLE_USER | Ein zweiter User ist nicht erlaubt. |
| DP_ ERROR_CI und<br>error->error_code ==<br>CI_RET_FL_NOT_CLEAR | Das Feld activated_fast_logic ist nicht Null. |
| Andere | Fehlerhafter Abschluss der Funktion. |

## 4.1.25 DP_fast_logic_off

### Zweck

Mit dieser Funktion kann ein Fast-Logic-Trigger des CP 5613/CP 5614 wieder deaktiviert werden.

### Syntax

```
DPR_DWORD DP_fast_logic_off (
                 DPR_DWORD         user_handle,      // in
                 DPR_WORD          fast_logic_id,    // in
                 DP_ERROR_T        *error);          // out
```

### Parameter

| Name | Beschreibung |
|---|---|
| user_handle | User-Handle, das beim Aufruf DP_open vergeben wurde. |
| fast_logic_id | Fast-Logic-Trigger<br>Wertebereich: 0 bis (DPR_MAX_FASTLOGIC_CNT - 1) - DPR_MAX_FASTLOGIC_CNT ist auf 4 festgelegt. |
| error | Adresse einer vom Anwenderprogramm bereitgestellten Struktur vom Typ DP_ERROR_T - Die Struktur enthält im Fehlerfall Details zur Fehlerursache (siehe Kapitel 4.4). |

### Return-Wert

| Name | Beschreibung |
|---|---|
| DP_OK | Der Fast-Logic-Trigger wurde deaktiviert. |
| DP_ ERROR_CI und<br>error->error_code ==<br>CI_RET_FL_ALREADY_OFF | Fast Logic ist bereits deaktiviert. |
| DP_ ERROR_CI und<br>error->error_code ==<br>CI_RET_FL_DOUBLE_USER | Ein zweiter User ist nicht erlaubt. |
| DP_ ERROR_CI und<br>error->error_code ==<br>CI_RET_FL_INV_ID | Parameter fast_logic_id ungültig. |
| DP_ ERROR_CI und<br>error->error_code ==<br>CI_RET_FL_NOT_CLEAR | Das Feld activated_fast_logic ist nicht Null. |
| Andere | Fehlerhafter Abschluss der Funktion. |

## 4.1.26 DP_watchdog

### Zweck

Diese Funktion startet oder stoppt auf dem CP 5613/CP 5614 einen Watchdog, welcher die Aktivität des DP-Master-Anwenderprogramms überwacht. Greift das DP-Master-Anwenderprogramm - nach Aufruf der Funktion - infolge eines Fehlverhaltens (z. B. Endlosschleife) nicht mehr auf das Watchdog-Element im Dualport RAM zu, wird dies vom CP 5613/CP 5614 nach Ablauf der Timeout-Zeit erkannt. Als Reaktion wechselt der CP daraufhin vom DP-Zustand OPERATE in den DP-Zustand CLEAR. Dadurch werden die Slaves in einen definierten Zustand gebracht.
Hinweise zu CLEAR siehe Kapitel 2.6 „Die Zustände des DP-Masters“



**Hinweis 1**

Die Aktivitätskontrolle kann verwendet werden, um gefährliche Anlagenzustände zu verhindern, die durch den Ausfall des DP-Master-Anwenderprogramms entstehen können.

Ohne Aktivitätskontrolle würde der DP-Master im DP-Zustand OPERATE weiterhin die letzten Ausgabedaten an die DP-Slaves senden, obwohl das DP-Master-Anwenderprogramm nicht mehr korrekt arbeitet. Ist ein solches Anlageverhalten nicht erwünscht, muss die Funktion DP_watchdog mit einer geeigneten der Anwendung angepassten Überwachungszeit aufgerufen werden.

**Hinweis 2**

Die Aktivitätskontrolle eines DP-Master-Anwenderprogramms ist nach DP_open oder nach DP_close ausgeschaltet. Wird im Parameter „timeout“ der Wert 0 übergeben, so wird eine gestartete Aktivitätskontrolle wieder beendet.

**Hinweis 3**

Der Vorgabewert 1 für die Überwachungszeit (10 ms) wird intern stets auf 2 (20 ms) aufgerundet. Die kleinste einstellbare Überwachungszeit beträgt somit 20 ms.

Die Genauigkeit der Überwachungszeit beträgt ca. 10 ms.

**Hinweis 4**

Ob der Watchdog abgelaufen ist, kann aus dem Strukturelement wd_state der Struktur DPR_WD_S entnommen werden (wd_state == DP_WD_TIMEOUT; siehe Kapitel 4.3.11 „User-Watchdog im Dualport RAM lesen und triggern“). Die Kennung DP_WD_TIMEOUT wird durch einen erneuten Aufruf von DP_watchdog oder durch DP_close wieder auf DP_WD_STOPPED zurückgesetzt.

**Hinweis 5**

Beim Multiapplikationsbetrieb wird jedem DP-Master-Anwenderprogramm, welches die Funktion DP_watchdog aufruft, ein eigener Watchdog zugeordnet. Läuft einer dieser Watchdogs ab, wird – unabhängig vom Zustand der anderen Watchdogs – der DP-Master selbsttätig vom Zustand OPERATE in den Zustand CLEAR heruntergefahren.

## Syntax

```
DPR_DWORD DP_watchdog (
               DPR_DWORD        user_handle,  // in
               DPR_DWORD        timeout,      // in
               DPR_WORD         *wd_index,    // out
               DP_ERROR_T       *error);      // out
```

## Parameter

| Name | Beschreibung |
|---|---|
| user_handle | User-Handle, das beim Aufruf DP_open vergeben wurde. |
| timeout | Überwachungszeit (in 10 Millisekunden).<br>**Werte:**<br>0: Deaktivieren der Aktivitätskontrolle<br>1 bis 65535: Einschalten der Aktivitätskontrolle |
| wd_index | Index auf eine Struktur DPR_WD_S innerhalb des Arrays user_watchdog im Dualport RAM. Der zurückgelieferte Index muss verwendet werden, um auf den Watchdog im Dualport RAM zuzugreifen. Nähere Informationen zum Triggern und Auslesen des Watchdog siehe Kapitel 4.3.11 „User-Watchdog im Dualport RAM lesen und triggern“. |
| error | Adresse einer vom Anwenderprogramm bereitgestellten Struktur vom Typ DP_ERROR_T - Die Struktur enthält im Fehlerfall Details zur Fehlerursache (siehe Kapitel 4.4). |

### Return-Wert

| Name | Beschreibung |
|---|---|
| DP_OK | Der Watchdog konnte aktiviert bzw. de-aktiviert werden |
| andere | Fehlerhafter Abschluss der Funktion. |

## 4.1.27 DP_write_trc

### Zweck

Diese Funktion ermöglicht es dem Anwenderprogramm, zu Test- und Debug-Zwecken einen selbstdefinierten Texteintrag in die DP-Trace-Datei zu schreiben. Optional ist es auch möglich, zusätzlich den Hex-Dump eines Speicherbereichs ausgeben zu lassen.

---

**Voraussetzung**

Damit der Trace-Eintrag in die Trace-Datei geschrieben wird, sind folgende Voraussetzungen erforderlich:

1. Aktivierung der DP-Trace-Datei durch das Programm „DP_Trace einstellen". Der Aufruf dieses Programms erfolgt über „PG/PC-Schnittstelle einstellen" (Diagnose -> erweiterte Diagnose -> DP_Trace).
2. Freischaltung der Funktion „DP_write_text" in der Trace-Auswahl des Programms „DP_Trace einstellen"
3. Übereinstimmung des Aufzeichnungs-Levels zum Vorgabewert in „DP_Trace einstellen". Der Aufzeichnungs-Level wird im Aufrufparameter trc_depth (siehe unten) gewählt.

---



**Hinweis 1**

Die DP-Trace-Datei beeinträchtigt die Ausführungsgeschwindigkeit und Dynamik des DP-Anwenderprogramms. Sie sollte daher nur zu Test- und Debug-Zwecken des DP-Anwenderprogramms aktiviert werden.

---

**Hinweis 2**

Der DP-Trace ist als Umlaufpuffer konzipiert. Daher werden alte Trace-Einträge überschrieben, wenn das Ende des Umlaufpuffers erreicht worden ist.

---

### Syntax

```
DPR_DWORD DP_write_trc (
     const DPR_STRING trc_text[DP_ERR_TXT_SIZE],// in
     DPR_WORD         trc_depth,                // in
     DPR_WORD         *trc_active,              // out
     const DPR_BYTE   *buf,                     // in
     DPR_BYTE         buf_len);                 // in
```

## Parameter

| Name | Beschreibung |
|---|---|
| trc_text | Ausgabe-String, welcher in die Trace-Datei geschrieben wird. Der String darf keine Escape-Sequenzen (Zeilenumbruch etc.) enthalten. |
| trc_depth | Mit diesem Parameter wird der Aufzeichnungs-Level des Trace-Eintrags festgelegt. Es kann der Wert DP_USR_TRC_DEPTH_MAX oder DP_USR_TRC_DEPTH_ERR eingetragen werden.<br>DP_USR_TRC_DEPTH_MAX: Es erfolgt immer ein Trace-Eintrag, sofern die obigen Voraussetzungen 1) und 2) erfüllt sind.<br>DP_USR_TRC_DEPTH_ERR: Es erfolgt nur dann ein Trace-Eintrag, wenn die obigen Voraussetzungen 1) und 2) erfüllt sind und zusätzlich die Option „nur Fehler und Ausnahmezustände“ in „DP_Trace einstellen“ gewählt worden ist. |
| trc_active | NULL oder Adresse einer vom Anwenderprogramm bereitgestellten Variablen. Nach Beendigung der Funktion enthält die Variable die Kennung, ob der DP-Trace aktiviert worden ist (1) oder nicht (0). Das DP-Anwenderprogramm kann mit Hilfe dieser Variablen zur Laufzeit entscheiden, ob Trace-Einträge generiert werden sollen. |
| buf | NULL oder Zeiger auf einen Datenbereich der Länge buf_len. Der Inhalt des Datenbereichs wird zusätzlich zum String trc_text als Speicher-Dump im Trace-File ausgegeben. |
| buf_len | Zahl der auszugebenden Bytes im durch buf adressierten Speicherbereich. |

## Return-Wert

| Name | Beschreibung |
|---|---|
| DP_OK | Trace-Aufruf ohne Fehler. Der Trace-Eintrag wird geschrieben, wenn die oben erwähnten Voraussetzungen erfüllt sind. |
| DP_ERROR_REQ_PAR | Parameterfehler |

## Beispiel für anwenderspezifische Trace-Ausgaben

Am Programmbeginn wird geprüft, ob der DP-Trace aktiviert ist. Es sei buf ein lokaler Puffer, der die Eingabedaten (4 Byte) des Slaves Nr. 5 enthält. Falls der DP-Trace aktiviert ist und Byte 0 ungleich 0 ist, soll in der Produktivphase zu Testzwecken ein DP-Trace-Eintrag erfolgen.

```
DPR_WORD IsTrcActive = 0;
/* Am Programmanfang pruefen, ob Trace aktiv ist */
if(DP_write_trc("Program Start",
                DP_USR_TRC_DEPTH_MAX,
                &IsTrcActive,
                NULL,
                0) != DP_OK)
{
  /* Parameterfehler!! */
}
                      ...
              /* Produktivphase */
/* Eingabedaten des Slaves Nr. 5 in buf einlesen */
/*             (siehe Kap. 4.3.1)               */

/* Trace-Ausgabe, falls DP_Trace aktiviert ist */
if (IsTrcActive)
{
  if(buf[0] != 0)
  {
    DP_write_trc("Slave 5: Input Bytes 0-3",
                 DP_USR_TRC_DEPTH_MAX,
                 NULL,
                 buf,
                 4);
  }
}
                      ....
```

## 4.2 Zusätzliche Funktionen des CP 5614

In diesem Kapitel sind die zusätzlichen Funktionen beschrieben, die für das Slave-Modul benötigt werden.

Die Funktionen des Slave-Moduls liegen in der „dps_base.dll", die Prototypen und Datenstrukturen sind in der Datei „dps_base.h" im Unterverzeichnis „prog" Ihrer Software-Installation abgelegt.

Funktionen, die mit „DP_" beginnen enthalten allgemeine und Master-Modulfunktionalität. Funktionen, die nur für das Slave-Modul relevant sind, beginnen mit „DPS_" (das S steht für Slave-Modul).

### 4.2.1 Übersichtstabellen zu den Slave-Modulfunktionen

#### Administrative Funktionen

In nachfolgender Tabelle sind auch die gemeinsam von Master- und Slave-Modul benutzten Funktionen, die mit DP_ beginnen, noch einmal aufgeführt.

| Name | Aufgabe |
|---|---|
| DP_start_cp<br>(Kap. 4.1.2) | Laden der Firmware und der Datenbasis in den CP 5614 - Benutzen Sie die Funktion des Masters. |
| DPS_open | Anmelden eines DPS-Anwenderprogramms, Vergabe eines User-Handles. |
| DP_get_pointer<br>(Kap. 4.1.5) | Zeiger auf das Prozessabbild anfordern - Benutzen Sie die Funktion des Masters. |
| DP_release_pointer<br>(Kap. 4.1.6) | Zeiger auf das Prozessabbild wieder zurückgeben - Benutzen Sie die Funktion des Masters. |
| DPS_close | Mit dieser Funktion meldet sich ein DPS-User wieder ab. |
| DP_reset_cp<br>(Kap. 4.1.3) | Anhalten der CP-Firmware - Benutzen Sie die Funktion des Masters. |
| DP_get_err_txt<br>(Kap. 4.1.8) | Diese Funktion gibt Fehlerinformationen im Klartext aus. Benutzen Sie die Funktion des Masters. |
| DP_init_sema_object<br>(Kap. 4.1.22) | Diese Funktion richtet ein Semaphor ein, an der Ihr Anwenderprogramm auf das Eintreffen von Ereignissen warten kann. |
| DP_del_sema_object<br>(Kap. 4.1.23) | Diese Funktion meldet ein Semaphor wieder ab. |

#### Initialisierungs-Funktionen

| Name | Aufgabe |
|---|---|
| DPS_start | Mit dieser Funktion wird das Slave-Modul aktiviert. |
| DPS_stop | Mit dieser Funktion wird das Slave-Modul abgeschaltet. Es reagiert dann nicht mehr am Bus |

## Standard-DP-Funktionen

| Name | Aufgabe |
|---|---|
| DPS_get_baud_rate | Diese Funktion ermittelt die aktuelle Datenübertragungsgeschwindigkeit. |
| DPS_get_gc_ command | Diese Funktion ermittelt das zuletzt empfangen Global-Control-Kommando. |
| DPS_get_state | Diese Funktion ermittelt den aktuellen Slave-Zustand. |
| DPS_set_diag | Diese Funktion setzt neue Diagnosedaten. |
| DPS_get_ind | Asynchrone Ereignisse empfangen. |
| DPS_set_resp | Asynchrone Ereignisse quittieren. |
| DPS_calc_io_data_len | I/O-Datenlänge anhand der Config-Daten berechnen. |

## 4.2.2 DPS_open

### Zweck

Mit dieser Funktion meldet sich ein DPS-Anwenderprogramm beim Treiber an und stellt die Slave-Parameter ein. Im Erfolgsfall gibt die Funktion ein User-Handle zurück. Das User-Handle muss bei allen weiteren Funktionsaufrufen verwendet werden.

---

**Hinweis 1**

Bei der Slave-Betriebsart DPS_SM_SIMPLE werden die Eingabedaten aus den Vorgabedaten für das Slave-Modul automatisch übernommen. Wenn der Slave nicht in der Betriebsart DPS_SM_SIMPLE betrieben wird, müssen nach der positiven Quittierung eines Konfigurationstelegramms die Eingabedaten des Slave-Moduls initialisiert (d. h. geschrieben) werden.

---

### Syntax

```
DPR_DWORD  DPS_open (
           const DPR_STRING  *cp_name,        // in
           DPR_DWORD         *user_handle,    // out
           DPR_DWORD         slave_mode,      // in
           DPR_WORD          station_addr,    // in
           DPR_WORD          addr_change,     // in
           DPR_WORD          pno_ident_nr,    // in
           DPR_WORD          user_wd,         // in
           DPS_INIT_DATA_T   *init_data,      // in
           DPS_MAX_DATA_T    *max_data,       // in
           DPR_WORD          baud_rate,       // in
           DP_ERROR_T        *error);         // out
```

## Parameter

| Name | Beschreibung |
|---|---|
| cp_name | Logischer Name eines CP 5614 (z. B. „CP_L2_1:“) |
| user_handle | Zeiger auf User-Handle-Variable - Im Erfolgsfall wird hier das dem Anwenderprogramm zugeordnete User-Handle eingetragen. |
| slave_mode | Einstellung der Slave-Betriebsart (die einzelnen Flags werden bitweise verknüpft):<br>• DPS_SM_SIMPLE — Einfacher Slave mit automatischer Prüfung von Parametrier- und Konfigurationsdaten.<br>• DPS_SM_V1_ENABLE — DP-V1-Dienste aktivieren<br>• DPS_SM_FREEZE_SUPP — Freeze unterstützen<br>• DPS_SM_SYNC_SUPP — Sync unterstützen<br>• DPS_SM_DYNAMIC — Parametrier- und Konfigurationsdaten des Slaves werden dynamisch geprüft. |
| station_addr | Stationsadresse des Slave |
| addr_change | 1 bedeutet: Adressänderung über Bus erlauben.<br>0 bedeutet: Adressänderung über Bus nicht erlauben. |
| pno_ident_nr | Von der PNO bei der Zertifizierung vergebene eindeutige Nummer für diesen Slave, z. B. 0x0008 für das Beispielanwenderprogramm. Diese Nummer wird im Intel-Format dargestellt. |
| user_wd | Rechenformel:<br>User_wd*10ms = Überwachungszeit des Anwenderprogramms<br>Wenn in dieser Zeit der Watchdog nicht nachgetriggert wird verlässt der Slave den Zustand Produktivbetrieb (Data_Ex). Mit 0 wird der Watchdog deaktiviert.<br>Ab welcher Software-Version diese Eigenschaft verfügbar ist, können Sie in der Versionstabelle im Kapitel 14.2 der Installationsanleitung nachlesen. Solange der Watchdog nicht realisiert ist, muss dieser Parameter mit 0 vorbelegt werden. |

Fortsetzung der Tabelle auf der nächsten Seite

Fortsetzung der Tabelle von der letzten Seite

<table>
<tr><th>Name</th><th>Beschreibung</th></tr>
<tr><td>init_data</td><td>Zeiger auf eine Struktur mit Informationen über die erweiterten Slave-Daten.<br>
<pre>
Typedef union DPS_INIT_DATA_S
{
     struct DPS_SIMPLE_S
     {
     DPR_WORD user_prm_data_len;   // in
     DPR_BYTE user_prm_data
           [DPS_MAX_PDU_LEN];      // in
     DPR_WORD cfg_data_len;        // in
     DPR_BYTE cfg_data[DPS_MAX_PDU_LEN];
                                   // in
     }simple;
     struct DPS_DYNAMIC_S
     {
     DPR_WORD def_cfg_data_len;    // in
     DPR_BYTE def_cfg_data
           [DPS_MAX_PDU_LEN];      // in
     }dynamic;
}DPS_INIT_DATA_T
</pre>
<table>
<tr><td>user_prm_data_len:</td><td>Länge der vorgegebenen User-Parametrierdaten <= 237 Byte</td></tr>
<tr><td>user_prm_data:</td><td>vorgegebene User-Parametrierdaten</td></tr>
<tr><td>cfg_data_len:</td><td>Länge der vorgegebenen Konfigurationsdaten <= 244 Byte</td></tr>
<tr><td>cfg_data:</td><td>vorgegebene vorgegebenen Konfigurationsdaten (Format siehe Kap. 4.7.3)</td></tr>
<tr><td>def_cfg_data_len:</td><td>Länge der Default-Konfigurationsdaten <= 244 Byte</td></tr>
<tr><td>def_cfg_data:</td><td>Default-Konfigurationsdaten</td></tr>
</table>
Anmerkung:<br>
Das erste User-Parametrierdaten-Byte ist für den Slave-Controller reserviert und wird bei der Überprüfung nicht berücksichtigt!<br>
Im DP-V1-Modus sind die ersten 3 User-Parametrierdatenbytes reserviert und werden bei der Überprüfung nicht berücksichtigt!</td></tr>
</table>

Fortsetzung der Tabelle auf der nächsten Seite

Fortsetzung der Tabelle von der letzten Seite

| Name | Beschreibung |
|---|---|
| max_data | Zeiger auf eine Struktur mit den maximalen Pufferlängen für Eingabe- und Ausgabedaten, für User-Diagnose, für User-Parametrier- und Konfigurationsdaten und zuletzt für die User-Set-Slave-Adressdaten.<br>`Typedef DPS_MAX_DATA_S`<br>`{`<br>`DPR_BYTE max_input_data_len;    // in`<br>`DPR_BYTE max_output_data_len;   // in`<br>`DPR_BYTE max_user_diag_len;     // in`<br>`DPR_BYTE max_user_prm_data_len; // in`<br>`DPR_BYTE max_cfg_data_len;      // in`<br>`DPR_BYTE max_user_ssa_data_len; // in`<br>`}DPS_MAX_DATA_T`<br>**Anmerkung:**<br>„max_user_ssa_data_len“ muss nicht ausgefüllt werden, falls addr_change mit 0 belegt ist.<br>Die Summe der Werte für max_input_data_len und max_output_data_len definiert die Länge des insgesamt zur Verfügung stehenden Datenpuffers für Ein- und Ausgabedaten. Eine max_input_data_len von 0 schließt somit bei genügend großer max_output_data_len den Empfang von Daten nicht aus. |
| baud_rate | Einzustellende Datenübertragungsgeschwindigkeit - Für den CP 5614 muss DPS_BD_AUTO_DETECT angegeben werden, da der CP 5614 über eine automatische Erkennung der Datenübertragungsgeschwindigkeit verfügt. |
| error | Adresse einer vom Anwenderprogramm bereitgestellten Struktur vom Typ DP_ERROR_T - Die Struktur enthält im Fehlerfall Details zur Fehlerursache (siehe Kapitel 4.4). |

## Return-Wert

| Name | Beschreibung |
|---|---|
| DP_OK | Erfolgreicher Abschluss der Funktion. |
| DP_ ERROR_CI und<br>error->error_code ==<br>CI_RET_OPEN_CP_NOT_<br>STARTED | CP ist nicht gestartet. |
| DP_ ERROR_EVENT_NET und<br>error->error_code ==<br>DPS_RET_DOUBLE_OPEN | DPS_open wurde bereits durchgeführt. |
| DP_ ERROR_EVENT_NET und<br>error->error_code ==<br>DPS_RET_NOT_OFFLINE | Das Slave-Modul ist nicht offline. |
| DP_ ERROR_REQ_PAR und<br>error->error_code ==<br>DPS_RET_INV_SLAVE_ADDR | Die angegebene Slave-Adresse ist un-<br>gültig. |
| DP_ ERROR_REQ_PAR und<br>error->error_code ==<br>DPS_RET_DIN_DOUT_LEN | Max. Input/Output falsch. |
| DP_ ERROR_REQ_PAR und<br>error->error_code ==<br>DPS_RET_DIAG_LEN | Max_user_diag_len falsch. |
| DP_ ERROR_REQ_PAR und<br>error->error_code ==<br>DPS_RET_PRM_LEN | Max_user_prm_data_len falsch. |
| DP_ ERROR_REQ_PAR und<br>error->error_code ==<br>DPS_RET_CFG_LEN | Max_cfg_data_len falsch. |
| DP_ ERROR_REQ_PAR und<br>error->error_code ==<br>DPS_RET_SSA_LEN | Max_user_ssa_data_len falsch. |
| DP_ ERROR_REQ_PAR und<br>error->error_code ==<br>DPS_RET_INV_CFG | Ungültige Default-Konfiguration. |
| DP_ ERROR_RES und<br>error->error_code ==<br>DP_RET_TOO_MANY_USR | Es können sich keine weiteren DP-<br>Instanzen beim CP anmelden. |
| DP_ ERROR_RES und<br>error->error_code ==<br>DPS_RET_NO_SLAVE_MODULE | DP-Slave-Funktionen sind nicht ver-<br>fügbar, weil es sich um einen CP 5613<br>handelt (ohne Slave-Modul). |
| DP_ ERROR_RES und<br>error->error_code ==<br>DPS_RET_LESS_MEM | Die angeforderten Puffer sind zu groß. |
| andere | Fehlerhafter Abschluss der Funktion. |

## 4.2.3 DPS_close

### Zweck

Mit dieser Funktion meldet sich ein DP-Anwenderprogramm wieder von der Kommunikation am Slave-Modul ab.

---

**Hinweis 1**

Nach dem erfolgreichen Abmelden ist das User-Handle nicht mehr gültig und darf nicht mehr weiter verwendet werden.

---

**Hinweis 2**

Um den DP-Slave herunterzufahren, sollte Ihr Anwenderprogramm vorher das Slave-Modul mit der Funktion DPS_stop in den Zustand OFFLINE bringen.

---

### Syntax

```
DPR_DWORD  DPS_close   (DPR_DWORD  user_handle,     // in
                        DP_ERROR_T  *error );        // out
```

### Parameter

| Name | Beschreibung |
| --- | --- |
| user_handle | User-Handle, das beim Aufruf DPS_open vergeben wurde. |
| error | Adresse einer vom Anwenderprogramm bereitgestellten Struktur vom Typ DP_ERROR_T. Die Struktur enthält im Fehlerfall Details zur Fehlerursache (siehe Kapitel 4.4). |

### Return-Wert

| Name | Beschreibung |
|---|---|
| DP_OK | Erfolgreicher Abschluss der Funktion |
| DP_ ERROR_EVENT_NET und error->error_code == DPS_RET_SEQUENCE_ERROR | Das Kommando DPS_close ist im momentanen Betriebszustand nicht erlaubt. |
| andere | Fehlerhafter Abschluss der Funktion |

## 4.2.4 DPS_start

### Zweck

Mit dieser Funktion kann der Slave ONLINE geschaltet werden. Das ist nach der Initialisierung nötig.

Nach einem OFFLINE-Schalten durch DPS_stop kann das Slave-Modul mit DPS_start wieder ONLINE geschaltet werden.

### Syntax

```
DPR_DWORD  DPS_start(
                     DPR_DWORD    user_handle,     // in
                     DP_ERROR_T   *error);         // out
```

### Parameter

| Name | Beschreibung |
|---|---|
| user_handle | User-Handle, das beim Aufruf DPS_open vergeben wurde. |
| error | Adresse einer vom Anwenderprogramm bereitgestellten Struktur vom Typ DP_ERROR_T. Die Struktur enthält im Fehlerfall Details zur Fehlerursache (siehe Kapitel 4.4). |

### Return-Wert

| Name | Beschreibung |
|---|---|
| DP_OK | Erfolgreicher Abschluss der Funktion |
| DP_ ERROR_EVENT_NET und error->error_code == DPS_RET_SEQUENCE_ERROR | Das Kommando DPS_start ist im momentanen Betriebszustand nicht erlaubt. |
| andere | Fehlerhafter Abschluss der Funktion |

## 4.2.5 DPS_stop

### Zweck

Diese Funktion dient dem Abschalten des Slave-Moduls. Das Slave-Modul reagiert dann nicht mehr am Bus.

### Syntax

```
DPR_DWORD  DPS_stop    (DPR_DWORD  user_handle,      // in
                         DP_ERROR_T *error );        // out
```

### Parameter

| Name | Beschreibung |
|---|---|
| user_handle | User-Handle, das beim Aufruf DPS_open vergeben wurde. |
| error | Adresse einer vom Anwenderprogramm bereitgestellten Struktur vom Typ DP_ERROR_T. Die Struktur enthält im Fehlerfall Details zur Fehlerursache (siehe Kapitel 4.4). |

### Return-Wert

| Name | Beschreibung |
|---|---|
| DP_OK | Erfolgreicher Abschluss der Funktion |
| DP_ ERROR_EVENT_NET und error->error_code == DPS_RET_SEQUENCE_ERROR | Das Kommando DPS_stop ist im momentanen Betriebszustand nicht erlaubt. |
| andere | Fehlerhafter Abschluss der Funktion |

### 4.2.6 DPS_get_baud_rate

#### Zweck

Mit dieser Funktion kann Ihr Anwenderprogramm die aktuelle Datenübertragungsgeschwindigkeit vom Slave-Modul abfragen.

#### Syntax

```
DPR_DWORD  DPS_get_baud_rate(
                      DPR_DWORD   user_handle      // in
                      DPR_WORD    *state,          // out
                      DPR_WORD    *baud_rate,      // out
                      DP_ERROR_T  *error );        // out
```

## Parameter

| Name | Beschreibung |
|---|---|
| user_handle | User-Handle, das beim Aufruf DPS_open vergeben wurde. |
| state | • DPS_BAUD_SEARCH: Keine Datenübertragungs-geschwindigkeit gefunden.<br>• DPS_BAUD_FOUND: Datenübertragungs-geschwindigkeit gefunden, Bus-Watchdog nicht aktiviert.<br>• DPS_BAUD_FOUND_WD: Datenübertragungs-geschwindigkeit gefunden, Bus-Watchdog aktiviert. |
| baud_rate | • DPS_BD_9K6 9600,00 kbit/s<br>• DPS_BD_19K2 19,20 kbit/s<br>• DPS_BD_45K45 45,45 kbit/s<br>• DPS_BD_93K75 93,75 kbit/s<br>• DPS_BD_187K5 187,5 kbit/s<br>• DPS_BD_500K 500,00 kbit/s<br>• DPS_BD_1M5 1,50 Mbit/s<br>• DPS_BD_3M 3,00 Mbit/s<br>• DPS_BD_6M 6,00 Mbit/s<br>• DPS_BD_12M 12,00 Mbit/s |
| error | Adresse einer vom Anwenderprogramm bereitgestellten Struktur vom Typ DP_ERROR_T. Die Struktur enthält im Fehlerfall Details zur Fehlerursache (siehe Kapitel 4.4). |

## Return-Wert

| Name | Beschreibung |
|---|---|
| DP_OK | Erfolgreicher Abschluss der Funktion |
| DP_ ERROR_RES und error->error_code == DPS_RET_NO_DPR_PTR | Kein Zugriff auf das DP-RAM möglich. |
| andere | Fehlerhafter Abschluss der Funktion |

## 4.2.7 DPS_get_gc_command

### Zweck

Diese Funktion ermittelt das zuletzt empfangene Global Control-Kommando, das vom Master, der diesen Slave steuert, gesendet wurde. Diese Funktion dient nur zur Information des Anwenderprogramms; es ist keine Reaktion des Anwenderprogramms notwendig um das Global-Control-Telegramm zu bearbeiten.

### Syntax

```
DPR_DWORD  DPS_get_gc_command(
                        DPR_DWORD   user_handle,      // in
                        DPR_WORD    *gc_cmd,          // out
                        DP_ERROR_T  *error);          // out
```

### Parameter

| Name | Beschreibung |
|---|---|
| user_handle | User-Handle, das beim Aufruf DPS_open vergeben wurde. |
| gc_cmd | Letztes empfangenes Global-Control-Kommando<br>Die Global-Control-Informationen sind in einzelnen Bits dargestellt, wobei mehrere Bits gleichzeitig gesetzt sein können. Der Aufbau entspricht dem „Global Control Byte“ der EN50170:<br>• DPS_CLEAR — Master ist im CLEAR-Zustand<br>• DPS_FREEZE — Eingänge werden übernommen und eingefroren.<br>• DPS_UNFREEZE — Eingänge werden wieder zyklisch aktualisiert.<br>• DPS_SYNC — Ausgänge werden einmalig aktualisiert.<br>• DPS_UNSYNC — Ausgänge werden wieder zyklisch aktualisiert. |
| error | Adresse einer vom Anwenderprogramm bereitgestellten Struktur vom Typ DP_ERROR_T. Die Struktur enthält im Fehlerfall Details zur Fehlerursache (siehe Kapitel 4.4). |

### Return-Wert

| Name | Beschreibung |
|---|---|
| DP_OK | Erfolgreicher Abschluss der Funktion |
| DP_ ERROR_RES und error->error_code == DPS_RET_NO_DPR_PTR | Kein Zugriff auf das DP-RAM mög-lich. |
| andere | Fehlerhafter Abschluss der Funktion |

### 4.2.8 DPS_get_state

#### Zweck

Diese Funktion ermittelt den Zustand des DP-Slave. Die Funktion dient nur zur Information der Anwenderprogramms.

---

**Hinweis 1**

Der Slave durchläuft während seines Betriebs die Zustände: OFFLINE, auf Parametrierdaten warten, auf Konfigurierdaten warten und zuletzt Datenaustausch.

Im Grundzustand ist er OFFLINE. Nach DPS_start erwartet er zuerst ein Parametriertelegramm (DPS_WAIT_PRM). Danach muss ein Konfigurationstelegramm kommen (DPS_WAIT_CFG). Werden diese beiden Telegramme als korrekt bestätigt, so geht der Slave in den Produktivbetrieb (DPS_DATA_EX).

---

**Hinweis 2**

Wenn der Slave nicht in der Betriebsart DPS_SM_SIMPLE betrieben wird, müssen nach der positiven Quittierung eines Konfigurationstelegramms die Eingabedaten des Slave-Moduls geschrieben werden. Erst nach einer Initialisierung der Eingabedaten kann der Slave in den Produktivbetrieb übergehen.

---

#### Syntax

```
DPR_DWORD  DPS_get_state(
                        DPR_DWORD   user_handle,     // in
                        DPR_WORD    *dps_state,      // out
                        DP_ERROR_T  *error);         // out
```

## Parameter

| Name | Beschreibung |
|---|---|
| user_handle | User-Handle, das beim Aufruf DPS_open vergeben wurde. |
| dps_state | • DPS_OFFLINE Das Slave-Modul ist nicht gestartet.<br>• DPS_WAIT_PRM Das Slave-Modul wartet auf ein Parametriertelegramm des remoten Masters.<br>• DPS_WAIT_CFG Das Slave-Modul wartet auf ein Konfigurationstelegramm des remoten Masters.<br>• DPS_DATA_EX Das Slave-Modul ist im Datenaustausch (Produktivphase). |
| error | Adresse einer vom Anwenderprogramm bereitgestellten Struktur vom Typ DP_ERROR_T - Die Struktur enthält im Fehlerfall Details zur Fehlerursache (siehe Kapitel 4.4). |

## Return-Wert

| Name | Beschreibung |
|---|---|
| DP_OK | Erfolgreicher Abschluss der Funktion |
| andere | Fehlerhafter Abschluss der Funktion |

## 4.2.9 DPS_set_diag

### Zweck

Mit dieser Funktion werden dem DP-Slave-Modul Diagnosedaten übergeben. Das CP 5614-Slave-Modul gibt diese Daten an den Master, der diesen Slave steuert, weiter.

---

**Hinweis**

Die Länge der Diagnosedaten kann während des Betriebs variieren!
Die 6 Byte Normdiagnose werden vom CP 5614-Slave-Modul selbst verwaltet.

Weitere Informationen bzgl. Diagnoseaufbau und Diagnoseformate siehe Kapitel 4.6.

---

### Syntax

```
DPR_DWORD  DPS_set_diag(DPR_DWORD  user_handle,      // in
                        DPR_BYTE   *user_diag_data,  // in
                        DPR_WORD   user_diag_len,    // in
                        DPR_WORD   diag_state,       // in
                        DP_ERROR_T *error );         // out
```

## Parameter

| Name | Beschreibung |
|---|---|
| user_handle | User-Handle, das beim Aufruf DPS_open vergeben wurde. |
| user_diag_data | Zeiger auf die User-Diagnosedaten ab dem 7. Byte - Die ersten 6 Bytes enthalten den Normteil, den die Slave-Modul-Hardware hinzufügt. (Format siehe Kap. 4.6). |
| user_diag_len | Länge der User-Diagnosedaten. |
| diag_state | Dieses Bitfeld kann aus folgenden Werten bitweise verknüpft werden:<br>• DPS_EXT_DIAG: Wenn gesetzt, handelt es sich um eine Fehlerinformation, ansonsten um eine Statusmeldung.<br>• DPS_EXT_DIAG_OV: es sind mehr Diagnosedaten vorhanden, als im Diagnosepuffer dargestellt werden können.<br>• DPS_STAT_DIAG: Es ist ein schwerwiegender Fehler aufgetreten, so dass keine sinnvollen Daten mehr geliefert werden können. |
| error | Adresse einer vom Anwenderprogramm bereitgestellten Struktur vom Typ DP_ERROR_T - Die Struktur enthält im Fehlerfall Details zur Fehlerursache (siehe Kapitel 4.4). |

## Return-Wert

| Name | Beschreibung |
|---|---|
| DP_OK | Erfolgreicher Abschluss der Funktion |
| DP_ ERROR_RES und error->error_code == DPS_RET_NO_DPR_PTR | Kein Zugriff auf das DP-RAM möglich. |
| andere | Fehlerhafter Abschluss der Funktion |

## 4.2.10 DPS_get_ind

### Zweck

Mit dieser Funktion kann eine Indication (Meldung vom steuernden Master) abgeholt werden.

---

**Hinweis 1**

Nennen Sie beim Aufruf die Indications, die Sie empfangen wollen. Sie erhalten dann genau eine Indication als Ergebnis zurück.

DPS_CHK_PRM wird auch gemeldet, wenn die User-Parametrierdatenlänge des remoten Masters = 0 ist! Somit hat das Slave-Modul die Chance auch eine Parametrierung ohne User-Daten abzulehnen.

---

**Hinweis 2**

Ist zum Zeitpunkt des Funktionsaufrufs keine Indication vorhanden und die Überwachungszeit abgelaufen, wartet die Funktion DPS_get_ind nicht, sondern liefert den Return-Wert DPS_NO_IND im Feld indication zurück.

Beim nächsten Aufruf kann neben der erwarteten Indication auch die vorher vergeblich erwartete Indication als Ergebnis zurückgegeben werden.

---

**Hinweis 3**

Wird beim Absetzen von DPS_close noch auf asynchrone Indications gewartet, wird DPS_get_ind mit der Fehlerklasse DP_ERROR_USR_ABORT beendet.

---

**Hinweis 4**

Dieser Aufruf darf zu einem Zeitpunkt von einem oder mehreren Programmen insgesamt nur einmal abgesetzt werden.

---

**Hinweis 5**

Ein Adresswechsel des Slave (DPS_NEW_SSA-Indication) kann nur durchgeführt werden, solange der Slave noch auf Parametrierdaten wartet. Nachdem ein Parametriertelegramm empfangen wurde, ist das Ändern der Slave-Adresse nicht mehr möglich.

---

## Syntax

```
DPR_DWORD  DPS_get_ind(
           DPR_DWORD      user_handle,          // in
           DPR_DWORD      *ind_ref,             // out
           DPR_DWORD      timeout,              // in
           DPR_DWORD      *indication,          // inout
           DPR_WORD       *data_len             // inout
           DPR_BYTE       *data_blk             // out
           DP_ERROR_T     *error );             // out
```

## Parameter

| Name | Beschreibung |
|---|---|
| user_handle | User-Handle, das beim Aufruf DPS_open vergeben wurde. |
| timeout | Dauer der maximalen Wartezeit (in Millisekunden), bis die Funktion zurückkehrt.<br>Grenzwerte:<br>0: keine Wartezeit (Funktion kehrt sofort zurück)<br>0x7FFFFFFE: maximale Wartezeit<br>DP_TIMEOUT_FOREVER: „unendliche“ Wartezeit |
| ind_ref | Adresse der Referenznummer der Indication - Muss bei DPS_set_resp wieder mit übergeben werden. |

Fortsetzung der Tabelle auf der nächsten Seite

Fortsetzung der Tabelle von der letzten Seite

| Name | Beschreibung |
|---|---|
| indication | Unter dieser Adresse trägt der Anwender die gewünschten Indications ein, über die er informiert werden möchte. Es handelt sich um Bits, die bitweise mit oder verknüpft werden können. Bei Rückkehr der Funktion wird der aktuelle Indication-Typ eingetragen (immer nur eine Indication)<br>• DPS_NO_IND: Keine Indication aufgetreten.<br>• DPS_CHK_PRM: Neues Parametriertelegramm wurde empfangen und muss von der Host-Software überprüft werden (nur für dynamischen Modus, wenn also beim DPS-open kein DPS_simple gewählt wurde).<br>• DPS_CHK_CFG: Neues Konfigurationstelegramm wurde empfangen und muss von Host-Software überprüft werden (nur für dynamischen Modus, wenn also beim DPS-open kein DPS_simple gewählt wurde).<br>• DPS_NEW_SSA: Set-Slave-Address –Telegramm wurde empfangen und muss von Host-Software überprüft werden (nur, wenn Adressänderung über Bus erlaubt ist).<br>• DPS_BAUD_CHANGED: Die Datenübertragungsgeschwindigkeitsinformation hat sich geändert (siehe DPS_get_baud_rate).<br>• DPS_GO_LEAVE_DATA_EX: Der Slave ist in den Zustand DPS_DATA_EX eingetreten oder hat diesen verlassen (siehe DPS_get_state).<br>• DPS_NEW_GC: Es wurde ein neues (geändertes) Global Control-Telegramm empfangen (siehe DPS_get_gc_ command). |

Fortsetzung der Tabelle auf der nächsten Seite

Fortsetzung der Tabelle von der letzten Seite

| Name | Beschreibung |
|---|---|
| data_len | maximale Länge des Arrays data_blk beim Aufruf der Funktion, Anzahl eingetragener Bytes nach Rückkehr der Funktion - Für den Aufruf sollte die Konstante DPS_MAX_PDU_LEN verwendet werden. |
| data_blk | Je nach Indication werden in diesem Array die Indication-Daten eingetragen (siehe nächster Absatz: Datenstruktur). |
| error | Adresse einer vom Anwenderprogramm bereitgestellten Struktur vom Typ DP_ERROR_T. Die Struktur enthält im Fehlerfall Details zur Fehlerursache (siehe Kapitel 4.4). |

## Datenstruktur (für Parameter data_blk)

Der Inhalt der Datenstruktur data_blk hängt vom Typ der Indication ab.

| Indication | Inhalt von Datenstruktur data_blk |
|---|---|
| DPS_CHK_PRM | `DPR_BYTE user_prm_data[ ];`<br>User-Parametrierdaten des Masters, ohne den Normanteil. |
| DPS_CHK_CFG | `DPR_BYTE cfg_data[ ];`<br>Konfigurationsdaten des Masters |
| DPS_NEW_SSA | `DPR_WORD new_address;`<br>`DPR_WORD ident_number;`<br>`DPR_WORD no_addr_chg;`<br>`DPR_BYTE user_data[ ];`<br>Datenstruktur, die die neue Stationsadresse, die Ident-Nummer, das Freischalten von Adressänderungen und die User-Daten enthält.<br>Alle Angaben sind im Intel-Format dargestellt. |
| DPS_BAUD_ CHANGED | `DPR_WORD state;`<br>`DPR_WORD baudrate;`<br>Datenstruktur, siehe DPS_get_baud_rate (siehe Kapitel 4.2.6). |
| DPS_GO_ LEAVE_DATA_ EX | `DPR_WORD dps_state;`<br>siehe Parameter DPS_get_state (siehe Kapitel 4.2.8). |
| DPS_NEW_GC | `DPR_WORD gc_command;`<br>siehe Parameter DPS_get_gc_command (siehe Kapitel 4.2.7). |

## Return-Wert

| Name | Beschreibung |
|---|---|
| DP_OK | Erfolgreicher Abschluss der Funktion |
| DP_ ERROR_EVENT_NET und<br>error->error_code ==<br>DPS_RET_SEQUENCE_ERROR | Das Kommando DPS_stop ist im momentanen Betriebszustand nicht erlaubt. |
| DP_ ERROR_REQ_PAR und<br>error->error_code ==<br>DPS_RET_BUF_LEN | Die Pufferlänge ist ungültig. |
| andere | Fehlerhafter Abschluss der Funktion |
| DP_ERROR_REQ_PAR und<br>code = DP_RET_PAR_USR_HANDL | User_handle falsch |
| DP_ERROR_REQ_PAR und<br>code = DP_RET_PAR_DATA_LEN | Parameter data_len falsch |

## 4.2.11 DPS_set_resp

### Zweck

Mit dieser Funktion stellt das Anwenderprogramm die Antwortdaten zu einer zuvor mittels DPS_get_ind empfangenen Indication zur Verfügung. Das ist für die Indications notwendig, die Informationen vom User benötigen, um die interne Bearbeitung zu steuern (DPS_CHK_PRM, DPS_CHK_CFG und DPS_NEW_SSA).

---

**Hinweis**

Die Indications DPS_BAUD_CHANGED, DPS_GO_LEAVE_DATA_EX, DPS_NEW_GC sowie DPS_NO_IND dürfen nicht quittiert werden!

---

### Syntax

```
DPR_DWORD  DPS_set_resp(
                       DPR_DWORD   user_handle,      // in
                       DPR_DWORD   ind_ref,          // in
                       DPR_WORD    data_len,         // in
                       DPR_BYTE    *data_blk,        // in
                       DP_ERROR_T  *error);          // out
```

### Parameter

| Name | Beschreibung |
|---|---|
| user_handle | User-Handle, das beim Aufruf DPS_open vergeben wurde. |
| ind_ref | Referenznummer, die von DPS_get_ind zurückgegeben wurde. |
| data_len | Datenlänge der Daten in data_blk |
| data_blk | Adresse der Antwortdaten |
| error | Adresse einer vom Anwenderprogramm bereitgestellten Struktur vom Typ DP_ERROR_T - Die Struktur enthält im Fehlerfall Details zur Fehlerursache (siehe Kapitel 4.4). |

## Datenstruktur (für Parameter data_blk)

<table>
<tr><th>Indication</th><th>Inhalt von Datenstruktur data_blk</th></tr>
<tr><td>DPS_CHK_PRM</td><td>DPR_WORD status
<table>
<tr><th>Wert</th><th>Bedeutung</th></tr>
<tr><td>DPS_PRM_OK</td><td>Parametrierungsdaten sind akzeptiert.</td></tr>
<tr><td>DPS_PRM_FAULT</td><td>Parametrierung wird nicht akzeptiert, Slave geht nicht in Datenaustausch.</td></tr>
</table></td></tr>
<tr><td>DPS_CHK_CFG</td><td>DPR_WORD status Formatierung
<table>
<tr><th>Wert</th><th>Bedeutung</th></tr>
<tr><td>DPS_CFG_OK</td><td>Konfigurationsdaten sind akzeptiert.</td></tr>
<tr><td>DPS_CFG_FAULT</td><td>Konfiguration wird nicht akzeptiert, Slave geht nicht in den Datenaustausch.</td></tr>
</table></td></tr>
<tr><td>DPS_NEW_SSA</td><td>DPR_WORD status Formatierung
<table>
<tr><th>Wert</th><th>Bedeutung</th></tr>
<tr><td>DPS_SSA_OK</td><td>Set-Slave-Address Bearbeitung abgeschlossen.</td></tr>
</table></td></tr>
</table>

## Return-Wert

| Name | Beschreibung |
|---|---|
| DP_OK | Erfolgreicher Abschluss der Funktion |
| DP_ERROR_EVENT_NET und error->error_code == DPS_RET_SEQUENCE_ERROR | Das Kommando ist im momentanen Betriebszustand nicht erlaubt. |
| andere | Fehlerhafter Abschluss der Funktion |
| DP_ERROR_REQ_PAR+ code = DP_RET_PAR_USR_HANDL | User_handle falsch |
| DP_ERROR_REQ_PAR+ code = DP_RET_PAR_DATA_LEN | Parameter data_len falsch |

### 4.2.12 DPS_calc_io_data_len

#### Zweck

Diese Funktion berechnet aus einem beliebigem Konfigurationstelegramm die Ein-/Ausgabedatenlänge. Die Funktion dient der Information des Anwenderprogramms, damit dieses nicht selbst die Kennungsbytes des Konfigurationstelegramms auswerten muss. Es kann ein beliebiges Konfigurationstelegramm ausgewertet werden.

#### Syntax

```
DPR_DWORD        DPS_calc_io_data_len (
                 DPR_WORD    cfg_len,                // in
                 DPR_BYTE    *cfg_data,              // in
                 DPR_WORD    *in_data_len,           // out
                 DPR_WORD    *out_data_len,          // out
                 DP_ERROR_T  *error );               // out
```

#### Parameter

| Name | Beschreibung |
|---|---|
| cfg_len | Länge der Konfigurationsdaten |
| cfg_data | Zeiger auf die Konfigurationsdaten |
| in_data_len | Zeiger auf die berechnete Input-Datenlänge |
| out_data_len | Zeiger auf die berechnete Output-Datenlänge |
| error | Adresse einer vom Anwenderprogramm bereitgestellten Struktur vom Typ DP_ERROR_T. Die Struktur enthält im Fehlerfall Details zur Fehlerursache (siehe Kapitel 4.4). |

#### Return-Wert

| Name | Beschreibung |
|---|---|
| DP_OK | Erfolgreicher Abschluss der Funktion |
| andere | Fehlerhafter Abschluss der Funktion |

## 4.3 Zugriffe auf das Prozessabbild des CP 5613/CP 5614

### Übersicht der vorhandenen Daten

Die folgende Tabelle gibt Ihnen eine Übersicht, welche Daten Ihrem Anwenderprogramm im Prozessabbild des CP 5613/CP 5614 zur Verfügung stehen.

| Kategorie | Daten |
|---|---|
| DP-Prozessdaten | • Eingabedaten der Slaves<br>• Ausgabedaten der Slaves<br>• Diagnosedaten der Slaves, inklusive Diagnosezähler<br>• Anzeige für Datenänderung der Slaves<br>• Eingabedaten des Slave-Moduls des CP 5614<br>• Ausgabedaten des Slave-Moduls des CP 5614 |
| Sonstige Informationen | • Information über den DP-Master<br>• Information über einen Slave<br>• Zustand der Slaves<br>• Aktuelle Busparameter<br>• Busstatistiken<br>• Watchdog des DP-Anwenderprogramms |
| Hardware-Events | • Steuerung der Event-Erzeugung bei Zyklusbeginn<br>• Steuerung der Event-Erzeugung bei Zyklusende<br>• Steuerung der Event-Erzeugung bei Änderung der Eingangsdaten der Slaves<br>• Steuerung der Event-Erzeugung bei Datenänderung des Slave-Moduls beim CP 5614<br>• Steuerung der Event-Erzeugung beim Empfang von Diagnosedaten der Slaves<br>• Abfrage des Fast-Logic-Status (zur Steuerung von Fast-Logic-Hardware-Events siehe die Kapitel 4.1.24 und 4.1.25) |

Die Formate der I/O-Daten und der Diagnosedaten werden in den Kapiteln 4.5 und 4.6 beschrieben.

## 4.3.1 Eingabedaten eines DP-Slave lesen

### Konsistentes Lesen

Für die Eingabedaten jedes einzelnen Slave gibt es einen fest zugeordneten Datenbereich im Prozessabbild des CP 5613/CP 5614.

Zum konsistenten Lesen der Eingabedaten eines Slave sperrt Ihr Anwenderprogramm diesen Datenbereich zunächst gegen Aktualisierung von Seiten des DP-Masters, greift dann darauf zu und gibt ihn anschließend wieder frei.

Das Sperren erfolgt durch das Schreiben der Slave-Nummer in ein Steuerregister im Prozessabbild. Das Freigeben erfolgt durch Schreiben des Wertes DPR_DP_UNLOCK oder einer anderen Slave-Nummer in das selbe Register.

### Beispiel für konsistentes Lesen

Es sei p ein Zeiger auf das Prozessabbild, den Ihr Anwenderprogramm beim „DP_get_pointer"-Aufruf erhalten hat. Dann werden 200 Byte des Slave Nr. 5 wie folgt in einen lokalen Puffer „buf" kopiert:

```
/* Sperren des Datenbereichs gegen Aktualisierung */
p->ctr.D_lock_in_slave_adr = 5;
/* Daten kopieren */
memcpy(buf, &p->pi.slave_in[5].data[0], 200);
/* Sperre wieder aufheben */
p->ctr.D_lock_in_slave_adr = DPR_DP_UNLOCK;
```



**Warnung 1**

Die Sperre wird auch aufgehoben durch:

- Sperren eines anderen Slave zum Lesen,
- Sperren eines Slaves zum Diagnose lesen (Kap. 4.3.2)
- Anstoßen eines Schreibvorgangs von Ausgangsdaten eines Slave über das zugehörige Steuerregister (Kap. 4.3.3)

**Hinweis 1**

Die Daten sind nur gültig, wenn der Master im Zustand OPERATE oder CLEAR/ AUTOCLEAR und der Slave im Zustand READY ist.

**Hinweis 2**

Der Speicherbereich mit Index 127 enthält die Daten des Slave-Moduls des CP 5614.

**Hinweis 3**

Die Eingabedaten werden nicht aktualisiert, solange der Slave statische Diagnose meldet (Kapitel 4.6.2, Byte 2 - Stationsstatus_2, Bit 1).

## 4.3.2 Diagnosedaten eines DP-Slave lesen

### Konsistentes Lesen der Diagnosedaten

Für die Diagnosedaten jedes einzelnen Slave gibt es einen fest zugeordneten Datenbereich im Prozessabbild des CP 5613/CP 5614 (zum Datenformat siehe Kapitel 4.6).

Zum konsistenten Lesen der Diagnosedaten eines Slave sperrt Ihr Anwenderprogramm den Datenbereich zunächst gegen Aktualisierung von Seiten des DP-Master, greift dann darauf zu und gibt ihn anschließend wieder frei. Das Sperren erfolgt durch das Schreiben der Slave-Nummer in ein Steuerregister im Prozessabbild. Das Freigeben erfolgt durch Schreiben des Wertes DPR_DP_UNLOCK in dasselbe Register.

Außerdem gibt es einen Zähler, der die Diagnosedaten des betreffenden Slave mitzählt.

### Beispiel für konsistentes Lesen der Diagnosedaten

Es sei p ein Zeiger auf das Prozessabbild, den Ihr Anwenderprogramm beim „DP_get_pointer“-Aufruf erhalten hat. Dann werden 200 Byte der Diagnosedaten des Slave Nr. 5 wie folgt in einen lokalen Puffer „buf“ kopiert:

```
/*Sperren des Diagnosedatenbereichs gegen Aktualisierung*/
p->ctr.D_lock_diag_slave_adr = 5;
/* Zähler auswerten */
count = p->pi.slave_diag[5].diag_count;
/* aktuelle Daten kopieren */
memcpy(buf, &p->pi.slave_diag[5].data[0],
                            p->pi.slave_diag[5].diag_len);
/* Sperre wieder aufheben */
p->ctr.D_lock_diag_slave_adr = DPR_DP_UNLOCK;
```

### Allgemeine Hinweise



**Warnung 1**

Die Sperre wird auch aufgehoben durch:

- Sperren eines Slaves zum Lesen dessen Eingangsdaten (Kap. 4.3.1)
- Sperren eines anderen Slave zum Lesen dessen Diagnose
- Anstoßen eines Schreibvorgangs von Ausgangsdaten eines Slave über das zugehörige Steuerregister (Kap. 4.3.3)



**Warnung 2**

Diagnosedaten dürfen nur konsistent gelesen werden.

**Hinweis 1**

Die Daten sind nur gültig, wenn der Master im Zustand STOP, OPERATE oder CLEAR und der Slave in der aktuellen Datenbasis projektiert ist.

**Hinweis 2**

Der Speicherbereich mit Index 127 enthält die Daten des Slave-Moduls des CP 5614.

**Hinweis 3**

Anhand des Diagnosezählers kann man erkennen, ob neue Diagnosedaten empfangen worden sind. Bei jeder eingehenden Diagnosemeldung wird der Diagnosezähler erhöht.

## 4.3.3 Ausgabedaten eines DP-Slave schreiben

### Schreiben geschieht immer konsistent

Für die Ausgabedaten jedes einzelnen Slave gibt es einen fest zugeordneten Datenbereich im Prozessabbild des CP 5613/CP 5614.

Zum konsistenten Schreiben der Ausgabedaten eines Slave schreibt Ihr Anwenderprogramm die Daten in den Datenbereich und stößt dann die Übernahme der Daten in den nächsten DP-Zyklus an, indem es die Slave-Nummer in ein Steuerregister im Prozessabbild des CP schreibt.

Aufgrund dieses Mechanismus des expliziten Anstoßen der Übertragung sind geschriebene Ausgangsdaten immer konsistent.

## Beispiel für konsistentes Schreiben

Es sei p ein Zeiger auf das Prozessabbild, den Ihr Anwenderprogramm beim „DP_get_pointer"-Aufruf erhalten hat. Dann werden 200 Byte im lokalen Puffer „buf" wie folgt an den Slave 5 geschrieben:

```
/* Daten kopieren */
memcpy(&p->pi.slave_out[5].data[0], buf, 200);
/* Übertragung anstoßen */
p->ctr. D_out_slave_adr = 5;
```



**Warnung**

Die Sperre wird auch aufgehoben durch:

- Sperren eines anderen Slave zum Lesen dessen Eingangsdaten (Kap. 4.3.1)
- Sperren eines anderen Slave zum Lesen dessen Diagnose

**Hinweis 1**

Die Daten werden nur zum Slave übertragen, wenn der Master im Zustand OPERATE und der Slave im Zustand READY ist.

**Hinweis 2**

Der Speicherbereich mit Index 127 enthält die Daten des Slave-Moduls des CP 5614.

**Hinweis 3**

Die Ausgabedaten werden nicht gesendet, solange der Slave statische Diagnose meldet (Kapitel 4.6.2, Byte 2 - Stationsstatus_2, Bit 1).

### 4.3.4 Prüfen der Slaves auf Datenänderung

#### Nutzen der Änderungsinformation

Im Prozessabbild des CP 5613/CP 5614 gibt es einen Speicherbereich, in dem Sie feststellen können, welche Slave-Daten sich geändert haben.
Diese Eigenschaft können Sie verwenden, um ohne Benutzung von Semaphore schnell feststellen zu können, wo sich Daten geändert haben.

#### Beispiel

Es sei p ein Zeiger auf das Prozessabbild, den Ihr Anwenderprogramm beim „DP_get_pointer"-Aufruf erhalten hat. Um z. B. zu zählen, bei wie vielen Slaves geänderte Daten vorliegen, würden Sie etwa folgendes programmieren, wobei „sum" das Ergebnis enthält:

```
Unsigned short sum = 0;
Unsigned short i;
for (i = 0; i< DPR_MAX_SLAVE_ADDR; i++)
{
   if ( p->ef.input[i].req_mask == DPR_DATA_CHANGE )
   {
      /* geänderte Daten beim Slave i gefunden */
      /* Sperren des Datenbereichs gegen Aktualisierung: */
      p->ctr.D_lock_in_slave_adr = i;

      /* Maske wieder freigeben: */
      p->ef.input[i].req_mask =
                               DPR_DATA_INT_CLEAR_AND_UNMASK;

      /* Daten lesen: */
      memcpy(...);

      /* Sperren des Datenbereichs aufheben: */
      p->ctr.D_lock_in_slave_adr = DPR_DP_UNLOCK;

      sum++;
   }
}
```

## Allgemeine Hinweise



**Warnung**

Ihr Anwenderprogramm muss die Eingangsdaten eines Slave zuerst sperren (D_lock_in_slave_addr, Kap. 4.3.1), bei Datenänderung das Feld „req_mask“ zurücksetzen (DPR_DATA_INT_CLEAR_AND_UNMASK bzw. DPR_DATA_INT_CLEAR_AND_MASK, siehe Hinweis 1) und dann die Daten lesen.

Sonst kann eine ungünstige Überlappung der Ereignisse dazu führen, dass die nächste Datenänderung nicht bemerkt wird.

**Hinweis 1**

Ihr Anwenderprogramm muss die Event-Maske selbst zurücksetzen. Dazu kann DPR_DATA_INT_CLEAR_AND_UNMASK (bei Benutzung von Semaphoren) oder DPR_DATA_INT_CLEAR_AND_MASK (Polling) benutzt werden (siehe dazu auch Kapitel 4.3.12 „Hardware-Event-Erzeugung ein- und ausschalten“.

**Hinweis 2**

Der Speicherbereich mit Index 127 enthält die Daten des Slave-Moduls des CP 5614.

### 4.3.5 Zustand eines DP-Slave feststellen

#### Nutzen des Slave-Zustands

Vor dem Zugriff auf Daten sollte Ihr Anwenderprogramm den Zustand eines Slave prüfen, um herauszubekommen, ob zu lesende Daten gültig sind.

## Beispiel

Es sei p ein Zeiger auf das Prozessabbild, den Ihr Anwenderprogramm beim „DP_get_pointer“-Aufruf erhalten hat. Um z. B. zu zählen, wieviele Slaves im Zustand READY sind, würden Sie etwa folgendes programmieren, wobei „sum“ das Ergebnis enthält:

```
unsigned short sum = 0;
unsigned short i;
for (i=0; i < DPR_MAX_SLAVE_ADDR;i++)
{
   if (p->info_watch.slave_info[i].slave_state ==
                                          DPR_SLV_READY)
   {
     sum++;  /* gefunden */
   }
}
```

Daneben gibt es noch den Zustand DPR_SLV_NOT_READY, der auch nicht projektierte Slaves einschließt.

---

**Hinweis 1**

Der DP-Master versucht automatisch, Slaves im Zustand NOT READY wieder neu zu initialisieren und in die zyklische Bearbeitung aufzunehmen.

---

**Hinweis 2**

Das Anwenderprogramm darf die Variable slave_state nicht überschreiben.

---

**Hinweis 3**

Der Speicherbereich mit Index 127 enthält den Zustand des Slave-Moduls des CP 5614.

---

**Hinweis 4**

Wenn der Slave im Zustand READY ist, bedeutet dies nicht zwangsweise, dass seine Daten gültig sind. Vorliegende Diagnosedaten können die Gültigkeit der Daten einschränken.

---

## 4.3.6 Informationen zum DP-Master abfragen

### Nutzen der DP-Master-Information

Im Prozessabbild des CP 5613/CP 5614 gibt es einen Speicherbereich, in dem Sie folgende Informationen über den DP-Master auslesen können:

- Zustand (OFFLINE, STOP, CLEAR, AUTOCLEAR, OPERATE)
- Ident-Nummer der Zertifizierung
- Hardware-Version
- Firmware-Version

### Beispiel

Es sei p ein Zeiger auf das Prozessabbild, den Ihr Anwenderprogramm beim „DP_get_pointer"-Aufruf erhalten hat. Um diese Informationen auf dem Bildschirm eines einfachen Anwenderprogramms auszugeben, würden Sie etwa folgendes programmieren:

```
printf("master state -> %ld",
                      p->info_watch.master_info.USIF_state);
     /* 0 bedeutet OFFLINE */
     /* 1 bedeutet STOP */
     /* 2 bedeutet CLEAR */
     /* 3 bedeutet AUTOCLEAR */
     /* 4 bedeutet OPERATE */
printf("master ident number -> %lx",
                    p->info_watch.master_info.ident_number);
     /* im Motorola-Format */
printf("master HW version -> %lx",
                      p->info_watch.master_info.hw_version);
     /* z. B. 0x00000102 für Version 1.2 */
printf("master FW version -> %lx",
                      p->info_watch.master_info.fw_version);
```

**Hinweis**

Ein Schreiben dieser Werte ist nicht zulässig und ändert die Master-Daten nicht.

## 4.3.7 Aktuelle Busparameter des Masters abfragen

### Nutzen der Busparameterabfrage

Im Prozessabbild des CP 5613/CP 5614 gibt es einen Speicherbereich, in dem Sie die aktuellen Busparameter auslesen können, um sie, z. B. durch Ihr Anwenderprogramm, anzeigen zu lassen.

---

**Hinweis**

Ein Schreiben dieser Werte ist nicht zulässig und ändert nicht die tatsächlich verwendeten Busparameter.

---

### Beispiel

Es sei p ein Zeiger auf das Prozessabbild, den Ihr Anwenderprogramm beim „DP_get_pointer“-Aufruf erhalten hat. Um diese Informationen auf dem Bildschirm eines einfachen Anwenderprogramms auszugeben, würden Sie etwa folgendes programmieren:

```
printf("Baudrate -> %d",
                    p->info_watch. aspc2_buspara.baud_rate);
```

Die Parameter sind Bytes, Worte oder Doppelworte. Lesen Sie die Formate ggf. im der Header-Datei nach, Struktur DPR_ASPC2_BUSPARA_T.

## Beschreibung der Busparameter

<table>
<tr><th>Name</th><th>Bedeutung</th></tr>
<tr><td>ts</td><td>Eigene Stationsadresse</td></tr>
<tr><td>baud_rate</td><td>Datenübertragungsgeschwindigkeit<br>
<table>
<tr><th>Wert</th><th>Bedeutung</th></tr>
<tr><td>DP_M_BAUDRATE_9K6</td><td>9,6 kbit/s</td></tr>
<tr><td>DP_M_BAUDRATE_19K2</td><td>19,2 kbit/s</td></tr>
<tr><td>DP_M_BAUDRATE_93K75</td><td>93,75 kbit/s</td></tr>
<tr><td>DP_M_BAUDRATE_187K5</td><td>187,5 kbit/s</td></tr>
<tr><td>DP_M_BAUDRATE_500K</td><td>500 kbit/s</td></tr>
<tr><td>DP_M_BAUDRATE_750K</td><td>750 kbit/s</td></tr>
<tr><td>DP_M_BAUDRATE_1M5</td><td>1,5 Mbit/s</td></tr>
<tr><td>DP_M_BAUDRATE_3M</td><td>3 Mbit/s</td></tr>
<tr><td>DP_M_BAUDRATE_6M</td><td>6 Mbit/s</td></tr>
<tr><td>DP_M_BAUDRATE_12M</td><td>12 Mbit/s</td></tr>
<tr><td>DP_M_BAUDRATE_31K25</td><td>31,25 kbit/s</td></tr>
<tr><td>DP_M_BAUDRATE_45K45</td><td>45,45 kbit/s</td></tr>
</table></td></tr>
<tr><td>tsl</td><td>Slot Time (in Bitzeiten)</td></tr>
<tr><td>min_tsdr</td><td>Minimum Station Delay (in Bitzeiten)</td></tr>
<tr><td>max_tsdr</td><td>Maximum Station Delay (in Bitzeiten)</td></tr>
<tr><td>tqui</td><td>Modulator-Ausklingzeit (in Bitzeiten)</td></tr>
<tr><td>tset</td><td>Setup-Time (in Bitzeiten)</td></tr>
<tr><td>ttr</td><td>Target Rotation Time (in Bitzeiten)</td></tr>
<tr><td>g</td><td>GAP-Aktualisierungsfaktor</td></tr>
<tr><td>hsa</td><td>Höchste PROFIBUS-Adresse</td></tr>
<tr><td>max_retry_limit</td><td>Max. Anzahl von Aufrufwiederholungen</td></tr>
<tr><td>station_type</td><td>1 (aktive Station); 0 (passive Station)</td></tr>
<tr><td>trdy</td><td>READY-Time (in Bitzeiten)</td></tr>
<tr><td>BpFlag</td><td>• Bit 7=0 bedeutet: keine Zustandswechsel im Fehlerfall<br>• Bit 7=1 bedeutet: Zustandswechsel im Fehlerfall (AUTOCLEAR)<br>• Bit 6 bis 0: reserviert</td></tr>
<tr><td>MinSlaveInterval</td><td>siehe Kapitel 3.2 (Einheit: 1 ms)</td></tr>
<tr><td>PollTimeout</td><td>Überwachungszeit der Kommunikation mit einem Master Klasse 2 (Einheit: 10 ms)</td></tr>
<tr><td>DataControlTime</td><td>siehe Kapitel 3.2 (Einheit: 1 ms)<br>Innerhalb dieser Zeitspanne teilt der DP-Master den zugeordneten Slaves seinen Betriebszustand mit.</td></tr>
</table>

## 4.3.8 Informationen zu DP-Slaves abfragen

### Nutzen der DP-Slave-Information

Im Prozessabbild des CP 5613/CP 5614 gibt es einen Speicherbereich, in dem Sie die projektierten Informationen über DP-Slaves auslesen können, um sie z. B. durch Ihr Anwenderprogramm anzuzeigen.

### Beispiel

Es sei p ein Zeiger auf das Prozessabbild, den Ihr Anwenderprogramm beim „DP_get_pointer"-Aufruf erhalten hat. Um diese Informationen auf dem Bildschirm eines einfachen Anwenderprogramms auszugeben, würden Sie etwa folgendes programmieren:

```
printf("Slave type -> %d",
                    p->info_watch.slave_info[5].slave_type);
     /* 0 bedeutet Slave ist nicht projektiert */
     /* 1 bedeutet Slave ist projektiert */
     /* 2 bedeutet DP-V1-fähiger Slave ist projektiert */
printf("Anzahl Ausgabebytes -> %d",
               p->info_watch.slave_info[5]. slave_out_byte);
printf("Anzahl Eingabebytes -> %d",
                p->info_watch.slave_info[5]. slave_in_byte);
printf("In Database -> %d",
            p->info_watch.slave_info[5]. slave_in_database);
```

## 4.3.9 PROFIBUS-Statistikdaten lesen

### Übersicht

Im Prozessabbild des CP 5613/CP 5614 gibt es einen Speicherbereich, in dem der CP Statistikdaten über den angeschlossenen PROFIBUS ablegt, damit z. B. Diagnoseprogramme lesend darauf zugreifen können.

### Beispiel für einen Zugriff

Es sei p ein Zeiger auf das Prozessabbild, den Ihr Anwenderprogramm beim „DP_get_pointer“-Aufruf erhalten hat. Um z. B. den Zähler der bisher festgestellten Buskurzschlüsse auszugeben, würden Sie etwa folgendes programmieren:

```
/* gebe Zähler Buskurzschlüsse aus */
printf("bus ctrl err =%u\n",
               p->info_watch.aspc2_event.bus_control_error);
```

## Beschreibung der verfügbaren Statistikzähler

Alle Elemente sind vorzeichenlose 16-Bit-Zähler, die das Auftreten des jeweiligen Ereignisses mitzählen. Die ersten vier Ereignisse sind besonders wichtig.

| Zählername | Bedeutung |
|---|---|
| off_ts_adr_error | Eine andere Station mit der gleichen Adresse wurde erkannt. |
| in_ring | Angabe, wie oft der CP in den Ring der aktiven PROFIBUS-Master eingetreten ist. |
| out_of_ring | Angabe, wie oft der CP aus dem Ring der aktiven PROFIBUS-Master herausgefallen ist (der CP ist momentan im Ring wenn in_ring > out_ring). |
| bus_control_error | Anzahl Buskurzschlüsse - Diese beeinträchtigen die Funktion des PROFIBUS und müssen beseitigt werden. |
| on_double_token | Angabe, wie oft Mehrfach-Token oder Token-Verlust erkannt wurde. |
| on_timeout | Angabe, wie oft der Token verloren ging und von diesem CP wieder erzeugt wurde. |
| on_syni_error | Angabe, wie oft sporadische Störungen auf der PROFIBUS-Leitung aufgetreten sind. |
| on_hsa_error | Angabe, wie oft eine Stationsadresse am Bus erkannt wurde, die höher ist, als die bei diesem CP projektierte HSA. |
| off_hsa_error | reserviert |
| on_response_error | Angabe, wie oft Fehler bei Response-Empfang auftraten. |
| on_las_useless | reserviert |
| on_rec_frame_ overflow | reserviert |
| on_fifo_error | reserviert |
| on_req_length_error | reserviert |
| off_pass_token_error | Angabe, wie oft der Token bei der Weitergabe gestört wurde. |

### 4.3.10 Fast-Logic-Status-Abfragen

#### Übersicht

Im Prozessabbild des CP 5613/CP 5614 gibt es einen Speicherbereich, in dem Sie nachsehen können, ob ein Fast-Logic-Trigger ausgelöst wurde.

DP-Master-Anwenderprogramm können das Fast-Logic-Semaphor (siehe Funktion DP_init_sema_object) benutzen, um sich asynchron über das Auslösen eines Fast-Logic-Triggers benachrichtigen zu lassen. Mit der Installation des Produkts wird ein Beispielprogramm mit installiert, welches die Funktionsweise der Fast Logic unter Verwendung des Fast-Logic-Semaphors demonstriert.

## Beispiel für die Abfrage eines Fast-Logic-Triggers

Es sei p ein Zeiger auf das Prozessabbild, den Ihr Anwenderprogramm beim „DP_get_pointer"-Aufruf erhalten hat. Um die Fast-Logic-Trigger von Ihrem Anwenderprogramm zu überwachen, würden Sie etwa folgendes programmieren:

```
/* Überprüfen, ob Fast-Logic-Trigger 0 aktiviert ist */
if(p->info_watch.activated_fast_logic[0] ==
     DP_FASTLOGIC_ACTIVATED)
{
 /*  Trigger ist aktiviert */
 /*  Zum Deaktivieren muss die Funktion
     DP_fast_logic_off()verwendet werden */

}
/* Überprüfen, ob Fast-Logic-Trigger 0 ausgelöst wurde */
if(p->info_watch.activated_fast_logic[0] ==
     DP_FASTLOGIC_TRIGGERED)
{
 /*  Trigger wurde aktiviert */
 /*  Trigger quittieren, damit er mit DP_fast_logic_on()
     wieder aktiviert werden kann */
 p->info_watch.activated_fast_logic[0] =
     DP_FASTLOGIC_CLEAR;
 /* Anwenderspezifische Aktionen */

}
```

**Hinweis 1**

In das Feld **p->info_watch.activated_fast_logic[i]** darf nur geschrieben werden, wenn **p->info_watch.activated_fast_logic[i]** den Wert **DP_FASTLOGIC_TRIGGERED** hat.

**Hinweis 2**

Es ist verboten das Feld **p->info_watch.activated_fast_logic[i]** mit einem anderen Wert als **DP_FASTLOGIC_CLEAR** zu besetzen.

## 4.3.11 User-Watchdog im Dualport RAM lesen und triggern

### Übersicht

Im Prozessabbild des CP 5613/CP 5614 gibt es einen Speicherbereich, in dem Sie den User-Watchdog nachtriggern und prüfen können. Der User-Watchdog wird durch den Aufruf DP_watchdog (siehe Kapitel 4.1.26 „DP_watchdog“) eingerichtet. Im Erfolgsfall gibt die Funktion einen Index zurück, der auf eine Watchdog-Struktur (DP_WD_S) innerhalb des Arrays user_watchdog im Dualport RAM zeigt.

Die Struktur DP_WD_S besteht aus den Elementen wd_state, wd_start, wd_counter und wd_trigger. Ein DP-Anwenderprogramm darf auf die Elemente wd_state, wd_start und wd_counter nur lesend zugreifen. Auf das Element wd_trigger darf das DP-Anwenderprogramm lesend und schreibend zugreifen.

| Strukturelement | Bedeutung |
|---|---|
| wd_state | Das Strukturelement wd_state zeigt den aktuellen Zustand des User-Watchdogs an. Es kann folgende Werte annehmen:<br>• DP_WD_STOPPED Watchdog ist deaktiviert.<br>• DP_WD_STARTED Watchdog ist gestartet, Aktivitätskontrolle läuft.<br>• DP_WD_TIMEOUT Watchdog ist abgelaufen, weil er nicht rechtzeitig vom DP-Anwenderprogramm getriggert worden ist. |
| wd_start | Das Strukturelement zeigt die aufgerundete Überwachungszeit, die beim Aufruf von DP_watchdog übergeben worden ist, in Einheiten von 10 Millisekunden an. |
| wd_counter | Das Strukturelement zeigt die bereits abgelaufene Überwachungszeit des Watchdog in Einheiten von 10 Millisekunden an. Es wird vom DP-Master zyklisch inkrementiert. |
| wd_trigger | Das Strukturelement dient zum Triggern des User-Watchdogs. Es muss vom DP-Master-Anwenderprogramm in zyklischen Abständen (d. h. mindestens 1-mal innerhalb der Watchdog-Überwachungszeit) inkrementiert werden. Durch das Triggern wird das Element wd_counter wieder auf den Wert 0 gesetzt und der Watchdog neu gestartet. Ein Überlauf des Wertebereichs von wd_trigger ist irrelevant und muss nicht überprüft werden. |

## Beispiel für einen Zugriff

Es sei p ein Zeiger auf das Prozessabbild, den Ihr Anwenderprogramm beim „DP_get_pointer“-Aufruf erhalten hat. Der Parameter idx sei der Wert, den die Funktion DP_watchdog als Index zurückgegeben hat. Um z. B. den aktuellen Zählerstand des Watchdogs auszulesen und den Watchdog zu triggern, würden Sie etwa folgendes programmieren:

```
/* gebe aktuellen Zählerstand des Watchdog aus */
printf("watchdog time =%u\n",
       p->info_watch.user_watchdog[idx].wd_counter);
/* Watchdog abgelaufen? */
if(p->info_watch.user_watchdog[idx].wd_state ==
                                            DP_WD_TIMEOUT)
{
  // Timeout -> user-spezifische Reaktion !!
}
else
{
  /* Watchdog triggern */
  p->info_watch.user_watchdog[idx].wd_trigger++;
}
```

### 4.3.12 Hardware-Event-Erzeugung ein- und ausschalten

#### Übersicht

Im Prozessabbild des CP 5613/CP 5614 gibt es einen Speicherbereich, der folgenden Hardware-Events zugeordnet ist:

- Beginn eines neuen DP-Zyklus (siehe Header-File DP_5613.H via D_cycle_start_mask und cycle_start_NT_PerformanceCounter)
- Ende des zyklischen Teils eines DP-Zyklus (siehe Header-File DP_5613.H via D_cycle_end_mask und cycle_end_NT_PerformanceCounter)
- Datenänderungen im DP-Prozessabbild
- Eintreffen von Diagnosedaten

Das Auslösen dieser Hardware-Events muss durch den Anwender aktiviert werden. Nach Eintreffen eines dieser Events, ist dieser automatisch deaktiviert.

Hardware-Events für Änderungen von Slave-Eingabedaten und Diagnose können für jeden Slave einzeln eingestellt werden.

Alle oben genannten Events werden durch das Weiterschalten von Semaphoren gemeldet.

## Beispiel für Aktivierung

Es sei p ein Zeiger auf das Prozessabbild, den Ihr Anwenderprogramm beim „DP_get_pointer“-Aufruf erhalten hat. Um Events bei Zyklusbeginn und bei Datenänderung und Diagnose von Slave 5 zu aktivieren, würden Sie etwa folgendes programmieren:

```
/* aktiviert Event bei Zyklusanfang */
p->ctr.D_cycle_start_mask = 0;
/* aktiviert Event bei Datenänderung von Slave 5 */
p->ef.input[5].req_mask = DPR_DATA_INT_CLEAR_AND_UNMASK;
/* aktiviert Event bei Eintreffen einer Diagnose
                                              von Slave 5 */
p->ef.diag[5].req_mask = DPR_DATA_INT_CLEAR_AND_UNMASK;
```

Anschließend werden beim Eintreffen der Events die von Ihrem Anwenderprogramm mit DP_init_sema_object angemeldeten Semaphore weitergeschaltet. Die anderen möglichen Werte von „req_mask“ sind:

- **DPR_DATA_INT_CLEAR_AND_MASK** – es wird auf einen Event gewartet, aber dann keine Semaphorweiterschaltung veranlasst, und
- **DPR_DATA_CHANGE** – ein Event ist eingetroffen.

## Beispiel für Deaktivierung

Obige Aktivierungssequenz würde Ihr Anwenderprogramm wie folgt rückgängig machen:

```
/* schaltet Event für Datenänderung von Slave 5 ab */
p->ef.input[5].req_mask = DPR_DATA_INT_CLEAR_AND_MASK;
/* schaltet Event bei Eintreffen einer Diagnose
                                          von Slave 5 ab */
p->ef.diag[5].req_mask = DPR_DATA_INT_CLEAR_AND_MASK;
```

**Hinweis 1**

Solange Ihr Anwenderprogramm ein Semaphor nicht durchlaufen hat, wird er nicht erneut aufgezogen. Sie sollten also beim Durchlaufen eines Semaphors stets prüfen, ob schon mehrere Events angekommen sind.

**Hinweis 2**

Nach Erhalt eines Hardware-Events wird die entsprechende Steuerbedingung zurückgesetzt, so dass Ihr Anwenderprogramm es erneut setzen muss.

Dadurch wird eine Überlastung des PC vermieden, wenn Ihr Anwenderprogramm die Hardware-Events nicht schnell genug bearbeiten sollte.

Wenn Sie eine Steuerbedingung erneut setzen, achten Sie bitte darauf, dass der Datenbereich des zugeordneten Slave gegen Aktualisierung gesperrt ist; Konsistentes Lesen (Kap. 4.3.1 bzw. Kap. 4.3.2).

**Hinweis 3**

Der Einsatz von Hardware-Events für viele aktive Slaves gleichzeitig belastet den PC stärker als Polling; siehe Ratschläge in der FAQ-Liste.

**Hinweis 4**

Der Hardware-Event bei Zyklusanfang und bei Ende des zyklischen Teils des DP-Zyklus kann durch den Anwender nicht deaktiviert werden.

### 4.3.13 Beim CP 5614 als DP-Slave Daten senden

#### Integration in das Prozessabbild des CP 5613/CP 5614

Die Sendedaten des Slave-Moduls liegen im Ausgabeabbild mit dem Slave-Index 127. Damit wird durch Beschreiben des Ausgabedatenbereichs mit dem Slave-Index 127 das Slave-Modul mit neuen zu sendenden Daten versorgt, die der übergeordnete Master als Eingänge liest.

#### Beispiel für konsistentes Schreiben

Es sei p ein Zeiger auf das Prozessabbild, den Ihr Anwenderprogramm beim „DP_get_pointer“-Aufruf erhalten hat. Dann werden 200 Byte im lokalen Puffer „buf“ wie folgt als Sendedaten ins Slave-Modul geschrieben:

```
/* Daten kopieren */
memcpy(&p->pi.slave_out[127].data[0], buf, 200);
/* Übertragung anstoßen */
p->ctr.–D_out_slave_adr = 127;
```

Die Daten werden jetzt ins Slave-Modul übertragen. Von dort kann sie der übergeordnete Master als Eingänge lesen.

---

**Hinweis 1**

Die Sendedaten werden vom Master nur übernommen, wenn das Slave-Modul des CP 5614 im Zustand READY und der Master im Zustand CLEAR, AUTOCLEAR oder OPERATE ist.

---

**Hinweis 2**

Auf die Zelle p->ctr.D_out_slave_adr darf nur schreibend zugegriffen werden.

---

### 4.3.14 Beim CP 5614 als DP-Slave Daten empfangen

#### Integration in das Prozessabbild des CP 5613/CP 5614

Die Empfangsdaten des Slave-Moduls werden im Eingabeabbild mit dem Slave-Index 127 abgelegt. Damit kann das Anwenderprogramm durch Lesen des Eingabedatenbereichs mit dem Slave-Index 127 die Daten lesen, die der übergeordnete Master als Ausgänge zum Slave-Modul gesendet hat.

#### Beispiel für konsistentes Lesen

Es sei p ein Zeiger auf das Prozessabbild, den Ihr Anwenderprogramm beim „DP_get_pointer"-Aufruf erhalten hat. Dann werden 200 Byte Empfangsdaten des Slave-Moduls wie folgt in einen lokalen Puffer „buf" kopiert:

```
/* Sperren des Datenbereichs gegen Aktualisierung */
p->ctr.D_lock_in_slave_adr = 127;
/* Daten wegkopieren */
memcpy(buf, &p->pi.slave_in[127].data[0], 200);
/* Sperre wieder aufheben */
p->ctr.D_lock_in_slave_adr = DPR_DP_UNLOCK;
```

Die kopierten Daten sind die Empfangsdaten des Slave-Moduls, die der übergeordnete Master als Ausgänge gesendet hat.

#### Lesen ohne Konsistenz

Beim Lesen ohne Konsistenz lassen Sie das Sperren und Freigeben des Datenbereichs einfach weg.

#### Allgemeine Hinweise

**Hinweis**

Die Daten sind nur gültig, wenn das Slave-Modul des CP 5614 im Zustand READY und der Master, der den Slave steuert, im Zustand OPERATE ist.

### 4.3.15 Beim CP 5614 als DP-Slave Diagnosedaten senden

#### Integration in CP 5613/CP 5614

Das Slave-Modul des CP 5614 kann seine Diagnosedaten an den übergeordneten Master weitergeben. Dazu wird nicht der Diagnosedatenbereich mit dem Slave-Index 127 verwendet, sondern eine C-Funktion.

Zum Format der Diagnosedaten siehe Kap. 4.6.

#### Beispiel für Diagnosedaten senden

Es sei h das User-Handle, das Ihr Anwenderprogramm beim DPS_open-Aufruf erhalten hat. Dann werden 10 Byte User-Diagnosedaten des Slave-Moduls wie folgt aus einem lokalen Puffer diag_buf gesendet:

```
/* Diagnosedaten an das Slave-Modul übergeben */
DPS_set_diag(h, diag_buf, 10, 0, &err);
```

**Hinweis**

Die Diagnosedaten werden vom Master nur übernommen, wenn das Slave-Modul des CP 5614 im Zustand READY und der Master, der den Slave steuert, im Zustand CLEAR, AUTOCLEAR oder OPERATE ist.

## 4.4 Fehlerbehandlung

### Einheitliche Fehlerstruktur DP_ERROR_T

Die Fehlerkennung für die einzelnen Aufträge benutzen die einheitliche Struktur DP_ERROR_T. Die verschiedenen Elemente geben im Fehlerfall die genaue Beschreibung der Fehlerursache zurück.

```
typedef struct DP_ERROR
{
     DPR_DWORD error_class;
     DPR_DWORD error_code;
     DPR_BYTE     error_decode;
     DPR_BYTE     error_code_1;
     DPR_BYTE     error_code_2;
} DP_ERROR_T;
```

## Strukturelement error_class

Das Strukturelement error_class gibt die allgemeine Fehlerklasse an. Der Eintrag in error_class ist **identisch** mit dem Rückgabewert des Funktionsaufrufs.

Ein DP-Anwenderprogramm kann daher alternativ das Strukturelement error_class oder den Rückgabewert des Funktionsaufrufs auswerten.

Die Tabelle beschreibt die möglichen Fehlerklassen und zeigt, welche der übrigen Strukturelemente von DP_ERROR_T bei den einzelnen Fehlerklassen gültig sind.

| Fehlerklasse | Beschreibung |
|---|---|
| DP_OK | Kein Fehler, der Auftrag wurde durchgeführt, die Ergebniswerte sind vorhanden. |
| DP_OK_ASYNC | Die Auftragsbearbeitung wurde erfolgreich angestoßen, die Confirmation ist noch nicht vorhanden. Das Ergebnis muss durch einen nachfolgenden DP_get_result-Aufruf abgeholt werden.<br>Dieses Ergebnis ist nur bei asynchronen Aufträgen möglich. |
| DP_ERROR_EVENT | Der Slave sendet bei einem der DPC1-Requests<br>• DP_ds_write<br>• DP_ds_read<br>• DP_alarm_ack<br>im Response-Telegramm eine Fehlercodierung zurück. |
| DP_ERROR_EVENT_NET | Fehler bei der PROFIBUS-Kommunikation, z. B. Verbindungsabbruch |
| DP_ERROR_REQ_PAR | Fehlerhafter Übergabeparameter eines Aufrufs oder Aufruf unzulässig. |
| DP_ERROR_CI | Fehler beim Zugriff auf den CP |
| DP_ERROR_RES | Nicht genügend Ressourcen vorhanden |
| DP_ERROR_USR_ABORT | Abbruch in Bearbeitung befindlicher Aufträge, weil das Anwenderprogramm sich abgemeldet hat (nur bei DP_get_result möglich). |

## Strukturelement error_code

Das Strukturelement error_code ist relevant bei den Fehlerklassen:

- DP_ERROR_EVENT_NET
- DP_ERROR_REQ_PAR
- DP_ERROR_CI
- DP_ERROR_RES
- DP_ERROR_USR_ABORT

Bei der Fehlerklasse DP_ERROR_EVENT_NET ist bei einigen Funktionsaufrufen zusätzlich das Strukturelement error_decode auszuwerten.

## Strukturelemente error_decode, error_code_1, error_code_2

Die Strukturelemente error_decode, error_code_1 und error_code_2 sind nur bei der Fehlerklasse DP_ERROR_EVENT relevant. Diese drei Elemente sind dabei immer gemeinsam auszuwerten. Sie enthalten die Fehlercodierung, die ein DP-V1-Slave nach einem DP-V1-Request im Antworttelegramm zurücksenden kann.

## Beschreibung der Fehlerklassen

Die folgende Abbildung 4 zeigt die prinzipielle Fehlerauswertung eines DP-Anwenderprogramms beim Aufruf einer DP-Funktion. <schaue Bild>

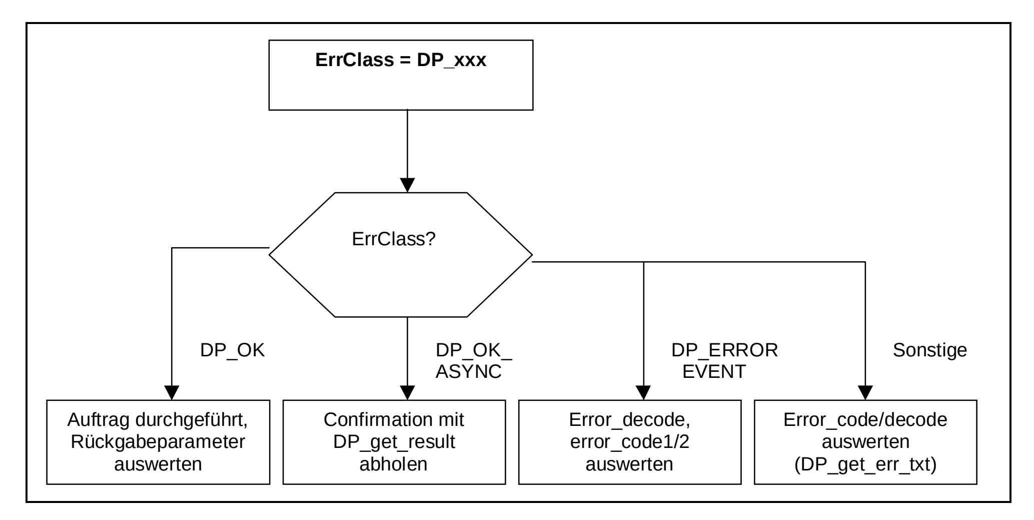



Abbildung 4 Auswertung der Rückgabewerte

## Bedeutung der einzelnen Fehler

Eine tabellarische Übersicht über die einzelnen Fehlercodes und ihre Einteilung in die oben beschriebenen Fehlerklassen bietet Kapitel 4.4.2 „Fehlercodes“. Zu jedem Fehlercode finden Sie eine kurze Beschreibung. Diese Beschreibung erhalten Sie auch durch den Aufruf der Funktion DP_get_err_txt, wenn der entsprechende Fehler aufgetreten ist.

### 4.4.1 Einträge in die Strukturelemente error_decode, error_code_1 und error_code_2 bei Fehlerklasse DP_ERROR EVENT

#### Beschreibung

Dieses Kapitel beschreibt die Einträge in die Strukturelemente:

- error_decode
- error_code_1
- error_code_2

**Alle drei Parameter sind dabei gemeinsam auszuwerten.**

Sie enthalten Fehlerinformationen, die vom DP-V1-Slave gesendet werden.

#### error_decode

Der Parameter error_decode legt die Bedeutung der Parameter error_code_1 und error_code_2 fest. Nachfolgende Tabelle zeigt den Wertebereich:

| Wertebereich | Bedeutung |
|---|---|
| 0 bis 127 | Reserviert |
| 128 | DP V1 |
| 129 bis 253 | Reserviert |
| 254 | PROFIBUS FMS |
| 255 | HART® |

## Error_Decode = PROFIBUS_FMS, HART®

Die Error_Decode-Werte PROFIBUS FMS und HART® zeigen an, dass die Parameter Error_Code_1 und Error_Code_2 wie bei den entsprechenden Protokollen definiert zu interpretieren sind.

**Bei PROFIBUS FMS:**

| Error_Code-Parameter | Bedeutung |
|---|---|
| Error_Code_1 | Error_**Class**_FMS |
| Error_Code_2 | Error_**Code**_FMS |

**Bei HART®:**

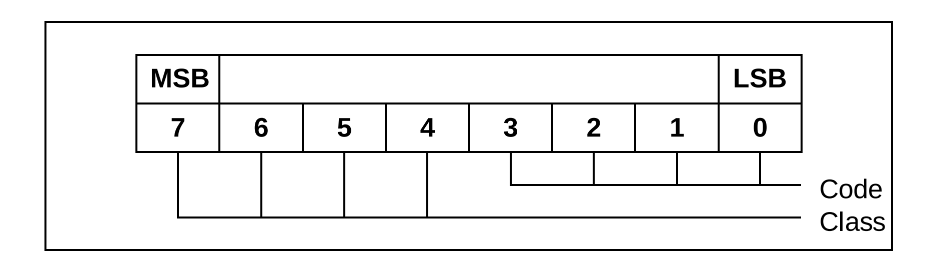
| Error_Code-Parameter | Bedeutung |
|---|---|
| Error_Code_1 | siehe HART-Spezifikation |
| Error_Code_2 | siehe HART-Spezifikation |

## Error_Decode = DP V1

Die Parameter Error_Code_1 und Error_Code_2 enthalten die DP-V1-spezifische Fehlerinformation.

## Aufbau von Error_Code_1 bei DP V1

Das Fehlerkodierungsschema von DP V1 verwendet Error_Code_1 für die Klassifizierung des Fehlers (Class) und für die Angabe eines zusätzlichen Fehler-Codes (Code).



## Error_Code_1

Der Wertebereich von Code und Class ist in nachfolgender Tabelle dargestellt:

| Byte-Struktur | | Bedeutung | |
|---|---|---|---|
| **Class** | **Code** | **Class** | **Code** |
| 0 bis 9 | beliebig | reserved | reserved |
| 10 | 0 | application | read error |
| | 1 | | write error |
| | 2 | | module failure |
| | 3 bis 7 | | reserved |
| | 8 | | version conflict |
| | 9 | | feature not supported |
| | 10 bis 15 | | user specific |
| 11 | 0 | access | invalid access |
| | 1 | | write length error |
| | 2 | | invalid slot |
| | 3 | | type conflict |
| | 4 | | invalid area |
| | 5 | | state conflict |
| | 6 | | access denied |
| | 7 | | invalid range |
| | 8 | | invalid parameter |
| | 9 | | invalid type |
| | 10 bis 15 | | user specific |
| 12 | 0 | resource | read constrain conflict |
| | 1 | | write constrain conflict |
| | 2 | | resource busy |
| | 3 | | resource unavailable |
| | 4 bis 7 | | reserved |
| | 8 bis 15 | | user specific |
| 13 bis 15 | Beliebig | user specific | user specific |

## Aufbau von Error_Code_2 bei DP V1

Der Parameter Error_Code_2 ist user-spezifisch.

## 4.4.2 Fehlercodes

### Übersicht

Dieses Kapitel enthält zu den Fehlerklassen

- DP_ERROR_EVENT_NET
- DP_ERROR_REQ_PAR
- DP_ERROR_CI
- DP_ERROR_RES

eine alphabetische Liste aller möglichen Fehlercodes sowie eine kurze Beschreibung zu jedem Fehlercode.

### Klasse DP_ERROR_EVENT_NET

| Klasse DP_ERROR_EVENT_NET Fehlercodes | Bedeutung |
|---|---|
| DP_RET_CP_ADR_NOT_IN_DB | Die Slave-Adresse ist nicht in der Datenbasis des CP enthalten. |
| DP_RET_CP_ ALARM_STATE_OVERFLOW | Fehler: Die maximale Anzahl der konfigurierten Alarme wurde vom Slave überschritten. |
| DP_RET_CP_ ALARM_STATE_INCONSISTENT | Fehler: Der Slave hat mehr als ein Alarm des gleichen Typs gesendet |
| DP_RET_CP_ ALARM_STATE_WRONG_TYPE | Der Slave hat einen Alarm gesendet, dessen Alarmtyp nicht konfiguriert ist. |
| DP_RET_CP_ ALARM_STATE_PDU_LENGTH | Der Slave hat einen Alarm oder eine Statusmeldung gesendet, deren Länge größer ist als spezifiziert worden ist. |
| DP_RET_CP_ ALARM_STATE_PDU_FORMAT | Der Slave hat Diagnosedaten mit Formatfehlern gesendet, so dass eine korrekte Dekodierung des Alarms oder des Status nicht möglich ist. |
| DP_RET_CP_CLOSED | Die Bearbeitungsinstanz in der Firmware wurde zuvor beendet und besteht nicht mehr. |
| DP_RET_CP_CONTROL_COMMAND | Control Command ist ungültig. |
| DP_RET_CP_DATABASE_ADR | Es befindet sich eine ungültige Slave-Adresse in der Datenbasis des CP. |
| DP_RET_CP_INIT_INSTANCE | Fehler beim DP Setup. |
| DP_RET_CP_L2_REQ | Unbekannter Opcode in der Confirmation. |

| **Klasse DP_ERROR_EVENT_NET Fehlercodes** | **Bedeutung** |
|---|---|
| DP_RET_CP_MEMORY_DPMC | Interner Speicherfehler im CP. |
| DP_RET_CP_NO_DATABASE | Es wurde keine (gültige) Datenbasis in den CP geladen. |
| DP_RET_CP_REQ_ACTIV | DP request schon in Bearbeitung. |
| DP_RET_CP_REQ_INVALID_LEN | Der Auftrag wurde mit Fehler beendet (ungültige Datenlänge). |
| DP_RET_CP_REQ_INVALID_PAR | Der Auftrag wurde mit Fehler beendet wegen ungültiger Parameter. |
| DP_RET_CP_REQ_NEG | Negativquittung beim Senden des Auftrags über den PROFIBUS.<br>Mögliche Ursachen: Slave antwortet nicht oder Dienstzugangspunkt beim Slave nicht aktiviert. |
| DP_RET_CP_REQ_NOT_ALLOWED | Der Dienst ist nicht erlaubt. |
| DP_RET_CP_REQ_NOT_FOUND | Der zugehörige Auftragsblock des Requests wurde nicht gefunden. |
| DP_RET_CP_REQ_RE | Format-Error in einem Response-Telegramm.<br>Quelle: lokale DP-Instanz (Direct Data Link Mapper). |
| DP_RET_CP_MM_FE | Format-Error in einem Request-Frame<br>Quelle: remote DP-Instanz (Direct Data Link Mapper) |
| DP_RET_CP_MM_NI | Funktion nicht implementiert<br>Quelle: remote User |
| DP_RET_CP_MM_AD | Zugang abgelehnt<br>Quelle: remote User |
| DP_RET_CP_MM_EA | Bereich zu groß (Up-/Download)<br>Quelle: remote User |
| DP_RET_CP_MM_LE | Datenblocklänge zu groß (Up-/Download)<br>Quelle: remote User |
| DP_RET_CP_MM_RE | Formatfehler in einem Response-Frame<br>Quelle: lokale DP-Instanz (Direct Data Link Mapper) |
| DP_RET_CP_MM_IP | Ungültiger Parameter<br>Quelle: remote User |
| DP_RET_CP_MM_SC | Sequenzkonflikt<br>Quelle: remote User |
| DP_RET_CP_MM_SE | Sequenzfehler<br>Quelle: remote DP-Instanz (Direct Data Link Mapper) |
| DP_RET_CP_MM_NE | Bereich existiert nicht<br>Quelle: remote User |

| Klasse DP_ERROR_EVENT_NET Fehlercodes | Bedeutung |
|---|---|
| DP_RET_CP_MM_DI | Daten unvollständig<br>Quelle: remote User |
| DP_RET_CP_MM_NC | Master Parametersatz nicht kompatibel<br>Quelle: remote User |
| DP_RET_CP_REQ_WITHDRAW | Der Auftrag wurde zurückgezogen und kann daher nicht bearbeitet werden. |
| DP_RET_CP_RESET_INSTANCE | Fehler beim DP-Reset. |
| DP_RET_CP_RESET_RUNNING | Reset ist bereits aktiviert. |
| DP_RET_CP_SET_MODE_CLR_ACT | Der DP_set_mode-Aufruf konnte nicht durchgeführt werden, weil ein voriger DP_set_mode-Aufruf (Wechsel in den Status Clear) noch in Bearbeitung ist. |
| DP_RET_CP_SET_MODE_FAIL | Fehler während der DP_set_mode-Bearbeitung. |
| DP_RET_CP_SET_MODE_OFFL_ACT | Der DP_set_mode-Aufruf konnte nicht durchgeführt werden, weil ein voriger DP_set_mode-Aufruf (Wechsel in den Status Offline) noch in Bearbeitung ist. |
| DP_RET_CP_SET_MODE_OPR_ACT | Der DP_set_mode-Aufruf konnte nicht durchgeführt werden, weil ein voriger DP_set_mode-Aufruf (Wechsel in den Status Operate) noch in Bearbeitung ist. |
| DP_RET_CP_SET_MODE_STOP_ACT | Der DP_set_mode-Aufruf konnte nicht durchgeführt werden, weil ein voriger DP_set_mode-Aufruf (Wechsel in den Status Stop) noch in Bearbeitung ist. |
| DP_RET_CP_SLV_NOT_ACTIV | Slave ist inaktiviert. |
| DP_RET_CP_SLV_NOT_IN_DATA | Der Slave ist (momentan) nicht bereit für den Datenaustausch. |
| DP_RET_CP_STARTED | Die Bearbeitungsinstanz wurde zuvor schon gestartet. |
| DP_RET_CP_STATE_UNKNOWN | Die Bearbeitungsinstanz ist in einem undefinierten Zustand. |
| DP_RET_CP_STOPPED | Die Bearbeitungsinstanz ist schon gestoppt worden. |
| DP_RET_CP_TIMEOUT | **Hinweis:** Der Auftrag wurde durch Timeout beendet. |
| DP_RET_CP_TIMER | Interner Timer-Fehler im CP. |
| DP_RET_CP_TOO_MANY_CTRL_CMD | Zu viele Global Control Commands sind in Bearbeitung. |
| DP_RET_CP_UNKNOWN_SLV_TYPE | Slave Typ unbekannt. |
| DP_RET_CP_USR_NOT_COMPATIBLE | Die dplib.dll und die dp_base.dll können sich nicht beim gleichen CP anmelden. |
| DP_RET_CP_WRONG_INSTANCE | Zugriff durch ungültige User Instanz. |

| Klasse DP_ERROR_EVENT_NET<br>Fehlercodes | Bedeutung |
|---|---|
| DP_RET_CP_WRONG_MODE_CLR | Ungültige Betriebsart beim Set Mode.<br>Aktuelle Betriebsart: DP_CLEAR<br>Erlaubte Betriebsart: DP_STOP, DP_OPERATE |
| DP_RET_CP_WRONG_MODE_OFL | Ungültige Betriebsart beim Set Mode.<br>Aktuelle Betriebsart: DP_OFFLINE<br>Erlaubte Betriebsart: DP_STOP |
| DP_RET_CP_WRONG_MODE_OPR | Ungültige Betriebsart beim Set Mode.<br>Aktuelle Betriebsart: DP_OPERATE<br>Erlaubte Betriebsart: DP_CLEAR |
| DP_RET_CP_WRONG_MODE_STP | Ungültige Betriebsart beim Set Mode.<br>Aktuelle Betriebsart: DP_STOP<br>Erlaubte Betriebsart: DP_OFFLINE, DP_CLEAR |
| DP_RET_REQ_ACTIV | Ein entsprechender Request ist schon in Bearbeitung. Erst nach Abholen der Quittung durch DP_get_result ist ein weiterer Request dieser Art möglich: Bei DP_ds_write, DP_ds_read und DP_get_actual_cfg ist einer dieser Requests sowie zusätzlich ein DP_alarm_ack pro Slave möglich.<br>Bei DP_enable_event können nicht mehrere Requests parallel bearbeitet werden. |
| DP_RET_TIMEOUT | **Hinweis:**<br>Die vorgegebene Timeout-Zeit ist abgelaufen.<br>Bedeutung bei DP_get_result: Es wurde keine Confirmation empfangen.<br>Bedeutung bei DP_get_pointer: Der Zeiger konnte nicht erhalten werden, Mögliche Ursache: weitere Applikationen haben den Zeiger in Benutzung. |

| Klasse DP_ERROR_EVENT_NET<br>Fehlercodes für die Slave-Modulfunktionen des CP 5614s | Bedeutung |
|---|---|
| DPS_RET_DOUBLE_OPEN | DPS_open wurde bereits durchgeführt. |
| DPS_RET_NOT_OFFLINE | Das Slave-Modul ist nicht offline |
| DPS_RET_SEQUENCE_ERROR | Das Kommando ist im momentanen Betriebszustand nicht erlaubt. |
| DPS_RET_UNKNOWN_ERROR | Unbekannter Fehler. |

## Klasse DP_ERROR_REQ_PAR

| Klasse DP_ERROR_REQ_PAR Fehlercodes | Bedeutung |
|---|---|
| DP_RET_CP_WRONG_FREEZE_GRP | Der Global Control-Aufruf (Freeze/Unfreeze) wurde mit Fehler beendet, weil der adressierte DP-Slave keiner der angegebenen Gruppe(n) zugehört. |
| DP_RET_CP_WRONG_GC_CMD | Der Auftrag wurde vom CP mit Fehler beendet (ungültiges Global Control Command). |
| DP_RET_CP_WRONG_GC_GRP | Der Auftrag wurde vom CP mit Fehler beendet (ungültige Global Control Gruppe). |
| DP_RET_CP_WRONG_SYNC_GRP | Der Global Control-Aufruf (Sync/Unsync) wurde mit Fehler beendet, weil der adressierte DP-Slave keiner der angegebenen Gruppe(n) zugehört. |
| DP_RET_PAR_ALARM | Der Parameter alarm ist ungültig.<br>Der Auftrag wurde von der DP_BASE-Library abgelehnt. |
| DP_RET_PAR_ALARM_TYPE | Der Parameter alarm_type ist ungültig.<br>Der Auftrag wurde von der DP_BASE-Library abgelehnt. |
| DP_RET_PAR_CP_NAME | Der Parameter cp_name ist ungültig.<br>Der Auftrag wurde von der DP_BASE-Library abgelehnt. |
| DP_RET_PAR_CREF | Der Parameter c_ref ist ungültig.<br>Der Auftrag wurde von der DP_BASE-Library abgelehnt. |
| DP_RET_PAR_CTRL_CMD | Der Parameter control_command ist ungültig.<br>Der Auftrag wurde von der DP_BASE-Library abgelehnt. |
| DP_RET_PAR_DATA | Der Parameter data ist ungültig.<br>Der Auftrag wurde von der DP_BASE-Library abgelehnt. |
| DP_RET_PAR_DATA_LEN | Der Parameter data_len ist ungültig.<br>Der Auftrag wurde von der DP_BASE-Library abgelehnt. |
| DP_RET_PAR_DPR | Der Parameter dpr (dual port ram) ist ungültig.<br>Der Auftrag wurde von der DP_BASE-Library abgelehnt. |
| DP_RET_PAR_FL | Der Parameter fast_logic ist ungültig.<br>Der Auftrag wurde von der DP_BASE-Library abgelehnt. |
| DP_RET_PAR_INDEX | Der Parameter index ist ungültig.<br>Der Auftrag wurde von der DP_BASE-Library abgelehnt. |
| DP_RET_PAR_LENGTH_M | Der Parameter length_m ist ungültig.<br>Der Auftrag wurde von der DP_BASE-Library abgelehnt. |
| DP_RET_PAR_LENGTH_S | Der Parameter length_s ist ungültig.<br>Der Auftrag wurde von der DP_BASE-Library abgelehnt. |
| DP_RET_PAR_MSG_FILTER | Der Parameter msg_filter ist ungültig.<br>Der Auftrag wurde von der DP_BASE-Library abgelehnt. |
| DP_RET_PAR_MST_MODE | Der Parameter mst_mode ist ungültig.<br>Der Auftrag wurde von der DP_BASE-Library abgelehnt. |

| Klasse DP_ERROR_REQ_PAR Fehlercodes | Bedeutung |
|---|---|
| DP_RET_PAR_REQ_TYPE | Der Parameter req_type ist ungültig.<br>Der Auftrag wurde von der DP_BASE-Library abgelehnt. |
| DP_RET_PAR_REQUEST | Der Parameter request ist ungültig.<br>Der Auftrag wurde von der DP_BASE-Library abgelehnt. |
| DP_RET_PAR_RESERVED | Der Parameter reserved ist ungültig.<br>Der Auftrag wurde von der DP_BASE-Library abgelehnt. |
| DP_RET_PAR_RESULT | Der Parameter result ist ungültig.<br>Der Auftrag wurde von der DP_BASE-Library abgelehnt. |
| DP_RET_PAR_SELECTOR | Der Parameter selector ist ungültig.<br>Der Auftrag wurde von der DP_BASE-Library abgelehnt. |
| DP_RET_PAR_SEMA_TYPE | Der Parameter sema_type ist ungültig.<br>Der Auftrag wurde von der DP_BASE-Library abgelehnt. |
| DP_RET_PAR_SLOT_NUMBER | Der Parameter slot_number ist ungültig.<br>Der Auftrag wurde von der DP_BASE-Library abgelehnt. |
| DP_RET_PAR_SLV_ADD | Der Parameter slv_add ist ungültig.<br>Der Auftrag wurde von der DP_BASE-Library abgelehnt. |
| DP_RET_PAR_SLV_MODE | Der Parameter slv_mode ist ungültig.<br>Der Auftrag wurde von der DP_BASE-Library abgelehnt. |
| DP_RET_PAR_TIMEOUT | Der Parameter timeout ist ungültig.<br>Der Auftrag wurde von der DP_BASE-Library abgelehnt. |
| DP_RET_PAR_TYPE | Der Parameter type ist ungültig.<br>Der Auftrag wurde von der DP_BASE-Library abgelehnt. |
| DP_RET_PAR_USR_HNDL | Der Parameter user_handle ist ungültig oder undefiniert.<br>Der Auftrag wurde von der DP_BASE-Library abgelehnt. |
| DP_RET_PAR_WD_INDEX | Der Parameter wd_index ist ungültig.<br>Der Auftrag wurde von der DP_BASE-Library abgelehnt. |

| **Klasse DP_ERROR_REQ_PAR**<br>**Fehlercodes für die Slave-Modulfunktionen des CP 5614** | **Bedeutung** |
|---|---|
| DPS_RET_BUF_LEN | Die Pufferlänge ist ungültig. |
| DPS_RET_CFG_LEN | Max_cfg_data_len falsch. |
| DPS_RET_DIAG_LEN | Max_user_diag_len falsch. |
| DPS_RET_DIN_DOUT_LEN | Max. Input/Output falsch. |
| DPS_RET_INV_CFG | Ungültige Default-Konfiguration. |
| DPS_RET_INV_SLAVE_ADDR | Parameter slave_addr falsch. |
| DPS_RET_INV_TIMEOUT | Der Parameter timeout ist ungültig. |
| DPS_RET_NOT_IMPLEMENTED | Der Dienst ist noch nicht implementiert. |
| DPS_RET_PAR_ASYNC_HDL | Der Parameter AsyncHandle ist ungültig. |
| DPS_RET_PAR_BAUD_RATE | Baud_rate falsch. |
| DPS_RET_PAR_CFG_DATA | Cfg_data falsch. |
| DPS_RET_PAR_CFG_LEN | Der Parameter cfg_len ist falsch. |
| DPS_RET_PAR_DATA_BLK | Data_blk falsch. |
| DPS_RET_PAR_DIAG_DATA | Diag_data falsch. |
| DPS_RET_PAR_DIAG_LEN | Diag_len falsch. |
| DPS_RET_PAR_DIAG_STATE | Diag_state falsch. |
| DPS_RET_PAR_GC | Gc falsch. |
| DPS_RET_PAR_IN_DATA_LEN | In_data_len falsch. |
| DPS_RET_PAR_IND_REF | Ind_ref falsch. |
| DPS_RET_PAR_INIT_DATA | Init_data falsch. |
| DPS_RET_PAR_OUT_DATA_LEN | Out_data_len falsch. |
| DPS_RET_PAR_STATE | State falsch. |
| DPS_RET_PAR_SYNC_HDL | Der Parameter SyncHandle ist ungültig. |
| DPS_RET_PRM_LEN | Max_user_prm_data_len falsch. |
| DPS_RET_SSA_LEN | Max_user_ssa_data_len falsch. |

## Klasse DP_ERROR_CI

| Klasse DP_ERROR_CI Fehlercodes | Bedeutung |
|---|---|
| CI_RET_ALREADY_CONNECTED | User hat schon Zugriff. |
| CI_RET_BLINK_INV_LED1_MS | Ungültige Zeit für Led1. |
| CI_RET_BLINK_INV_LED2_MS | Ungültige Zeit für Led2. |
| CI_RET_BLINK_INV_MODE | Ungültiger Mode. |
| CI_RET_BUF_NOT_VALID | Ungültiger Userbuffer. |
| CI_RET_CANCEL_NO_RECEIVE | Hinweis:<br>Nichts da zum abbrechen. |
| CI_RET_CIB_CLOSE_ALREADY | Doppelter Close. |
| CI_RET_CIB_HOST_READY | Datensemaphore hat keinen plausiblen Zustand. |
| CI_RET_CIB_INV_FILL_LENGTH_1 | Fill_length ist ungültig. |
| CI_RET_CIB_INV_NEXT_BLOCK | Next_block ist ungültig. |
| CI_RET_CIB_INV_NEXT_REQUEST | Next_request ist ungültig. |
| CI_RET_CIB_MAX_INDEX | Index ungültig. |
| CI_RET_CIB_NEXT_BLOCK | Request-Verkettung fehlerhaft. |
| CI_RET_CIB_NEXT_INDEX | Request-Verkettung fehlerhaft. |
| CI_RET_CIB_OPEN_ALREADY | Doppelter Open. |
| CI_RET_CIB_OPEN_HANDLE | Ungültiges Handle. |
| CI_RET_CIB_SUB_NOT_IMP | Subsystem nicht implementiert. |
| CI_RET_CIB_SUBSYSTEM | Subsystem ist ungültig. |
| CI_RET_CONNECT_DOWN | Interner Fehler. |
| CI_RET_CONNECT_DPR | Interner Fehler. |
| CI_RET_CONNECT_MAX | Maximale CI_connect_cp Anzahl überschritten. |
| CI_RET_CONNECT_NO_CP | Interner Fehler. |
| CI_RET_CONNECT_PLX | Interner Fehler. |
| CI_RET_CONNECT_TWICE | Interner Fehler. |
| CI_RET_CP_NAME_00 | Interner Fehler. |
| CI_RET_CP_NAME_NOT_FOUND | Ungültiger CP Name. |
| CI_RET_CP_NAME_TOO_LONG | CP Name ist zu lang. |
| CI_RET_CP_NOT_HERE | CP nicht vorhanden. |
| CI_RET_CP561X_ROOT | Eintrag fehlt in der Registry -> Fehler bei der Installation. |
| CI_RET_DISCONNECT_NOT_CONNECTED_0 | Kein mapping angelegt -> interner Fehler. |
| CI_RET_DISCONNECT_NOT_CONNECTED_1 | Kein mapping angelegt -> interner Fehler. |
| CI_RET_DISCONNECT_NOT_CONNECTED_2 | Kein mapping angelegt -> interner Fehler. |
| CI_RET_DUMP_FILENAME_TOO_LONG | File Name zu lang. |

| Klasse DP_ERROR_CI<br>Fehlercodes | Bedeutung |
|---|---|
| CI_RET_DUMP_OPEN_FILE | Das File für den DPR Abzug kann nicht geöffnet werden. |
| CI_RET_EXCP_ALERTED<br>CI_RET_EXCP_APPEND_0<br>CI_RET_EXCP_APPEND_1<br>CI_RET_EXCP_CIB<br>CI_RET_EXCP_CLOSE_0<br>CI_RET_EXCP_CLOSE_1<br>CI_RET_EXCP_CRITICAL_0<br>CI_RET_EXCP_CYCLE_0<br>CI_RET_EXCP_CYCLE_1<br>CI_RET_EXCP_E_LIST_0<br>CI_RET_EXCP_E_LIST_1<br>CI_RET_EXCP_GET_DP_ACCESS<br>CI_RET_EXCP_KE_WAIT<br>CI_RET_EXCP_LIFE_COUNTER<br>CI_RET_EXCP_OPEN<br>CI_RET_EXCP_QUEUE_DPA_0<br>CI_RET_EXCP_QUEUE_DPA_1<br>CI_RET_EXCP_REL_DP_ACCESS_0<br>CI_RET_EXCP_REL_DP_ACCESS_1<br>CI_RET_EXCP_RETURN_0<br>CI_RET_EXCP_RETURN_1<br>CI_RET_EXCP_RETURN_2<br>CI_RET_EXCP_USER_APC | Exceptions -> Interne Fehlerzustände des Treibers oder der Firmware.<br>Abhilfe -> CP zurücksetzen oder den PC neu booten. |
| CI_RET_FL_ALREADY_OFF | Fast Logic schon deaktiviert. |
| CI_RET_FL_ALREADY_ON | Fast Logic schon aktiviert. |
| CI_RET_FL_DOUBLE_USER | Ein zweiter User ist nicht erlaubt. |
| CI_RET_FL_INV_ACTION | Interner Fehler. |
| CI_RET_FL_INV_ADDR_IN_BYTE | Ungültige Adresse des Eingabe-Slaves. |
| CI_RET_FL_INV_ADDR_OUT_BYTE | Ungültige Adresse des Ausgabe-Slaves. |
| CI_RET_FL_INV_ID | Ungültiger Parameter fast_logic_id -> 0..3. |
| CI_RET_FL_INV_IN_MASK | Ungültige Maske für das Eingabe Byte. |
| CI_RET_FL_INV_INDEX_IN_BYTE | Ungültiges Eingabebyte. |
| CI_RET_FL_INV_INDEX_OUT_BYTE | Ungültiges Ausgabebyte. |
| CI_RET_FL_INV_OUT_MASK | Ungültige Maske für das Ausgabe Byte. |
| CI_RET_FL_NOT_CLEAR | Feld activated_fast_logic[id] ist nicht Null. |

| **Klasse DP_ERROR_CI**<br>**Fehlercodes** | **Bedeutung** |
|---|---|
| CI_RET_FL_SLAVE_IN_NOT_IN_DB | Eingabe-Slave nicht in der Datenbasis. |
| CI_RET_FL_SLAVE_OUT_NOT_IN_DB | Ausgabe-Slave nicht in der Datenbasis. |
| CI_RET_GET_CFG_INV_NAME | Interner Fehler. |
| CI_RET_GET_CFG_WRONG_NAME | Interner Fehler. |
| CI_RET_GET_MAX_PENDING | Maximale Anzahl pro CI_open überschritten. |
| CI_RET_GET_TIMEOUT | Hinweis:<br>Timeout abgelaufen. |
| CI_RET_HANDLE_CLOSING | Handle wird gerade geschlossen. |
| CI_RET_HANDLE_CP_RESET | Der CP wurde zurückgesetzt -> Beenden Sie die Applikation. |
| CI_RET_HANDLE_INVALID | Ungültiger Wert für User Handle. |
| CI_RET_HANDLE_NOT_OPEN | Handle nicht bekannt. |
| CI_RET_INV_SEMA_TYPE | Ungültiger sema type. |
| CI_RET_INV_USR_BUF_FILL_LENGTH | Fill_length > buf_length. |
| CI_RET_INV_USR_BUF_LENGTH | Puffer zu lang. |
| CI_RET_INV_USR_OPCODE | Ungültiges Subsystem. |
| CI_RET_NO_CP_NAME | Bei der Funktion muss der CP Name angegeben werden. |
| CI_RET_NO_DRIVER | Treiber nicht geladen. |
| CI_RET_NOT_IMPLEMENTED | Funktion nicht implementiert. |
| CI_RET_OPEN_CP_NOT_STARTED | CP nicht gestartet. |
| CI_RET_OPEN_MAX | Maximale CI_open Anzahl überschritten. |
| CI_RET_OPEN_REG | Interner Fehler. |
| CI_RET_OPEN_TEMP_LOCKED | Dienst momentan nicht möglich. |
| CI_RET_RCV_EVENT_INV_OPC | Interner Fehler. |
| CI_RET_RECEIVE_BUF_TOO_SMALL_0 | User Buffer ist zu klein. |
| CI_RET_RECEIVE_BUF_TOO_SMALL_1 | |
| CI_RET_RECEIVE_MAX_PENDING | Zu viele Receive mit Timeout pro CI_open. |
| CI_RET_RECEIVE_TIMEOUT_CANCEL | Hinweis:<br>Receive wurde abgebrochen. |
| CI_RET_RECEIVE_TIMEOUT_NO_DATA | Timeout abgelaufen. |
| CI_RET_RECEIVE_TIMEOUT_USER_APC | Interner Fehler. |
| CI_RET_RELEASE_NO_ACCESS | User hat keinen Zugriff gehabt. |
| CI_RET_RESET_ALREADY_DONE | Hinweis:<br>CP ist schon im Zustand Reset. |

| **Klasse DP_ERROR_CI Fehlercodes** | **Bedeutung** |
|---|---|
| CI_RET_RESET_CP_NOT_FOUND | CP wurde nicht gefunden. |
| CI_RET_RESET_CP_USED | Reset momentan nicht möglich. |
| CI_RET_RESET_INVALID_CP_NAME | Ungültiger CP Name. |
| CI_RET_RESET_INVALID_MODE | Ungültiger Reset Mode. |
| CI_RET_SCP_BUF_FILL_LENGTH_INV | Buf_fill_length invalid. |
| CI_RET_SCP_FILL_LENGTH_1_TOO_BIG | Fill_length_1 > seg_length_1. |
| CI_RET_SCP_FILL_LENGTH_2_TOO_BIG | Fill_length_2 > seg_length. |
| CI_RET_SCP_OFFSET_1_INVALID | Offset_1 != 80. |
| CI_RET_SCP_OFFSET_2_INVALID | Offset_2 out of range. |
| CI_RET_SCP_OPEN_MAX | Maximale Anzahl der SCP_open erreicht. |
| CI_RET_SCP_PARAM | Ungültiger Parameter. |
| CI_RET_SCP_SEG_LENGTH_1_TOO_BIG | Seg_length_1 > 260. |
| CI_RET_SCP_SEG_LENGTH_2_TOO_BIG | Seg_length_2 > 260. |
| CI_RET_SCP_WRONG_SUBSYSTEM | Subsystem ist nicht 0x22. |
| CI_RET_SEMA_NOT_INITIALIZED | Semaphore nicht initialisiert. |
| CI_RET_SEMA_TWICE | Für das User Handle existiert schon ein Objekt von diesem Typ. |
| CI_RET_SEND_BUF_TOO_SMALL | User Buffer für die Antwort ist zu klein. |
| CI_RET_SEND_NO_BUFFER_AVAILABLE | Ressourcen Engpass im DPR. |
| CI_RET_SET_HWND_MSG | Fehler bei der Funktion SetSinecHWndMsg. |
| CI_RET_START_ACTION_ERR | Interner Fehler. |
| CI_RET_START_ALREADY_DATABASE | Zweiter Start mit einer unterschiedlichen Datenbasis. |
| CI_RET_START_ALREADY_DONE | Hinweis:<br>CP schon gestartet. |
| CI_RET_START_CP_NO_REACTION | Interner Fehler -> CP reagiert nicht. |
| CI_RET_START_CP_NOT_FOUND | CP wurde nicht gefunden. |
| CI_RET_START_CP_RESOURCES_INT | Interner Fehler -> Ressourcen-Problem. |
| CI_RET_START_DOWN_1 | Interner Fehler. |
| CI_RET_START_ERROR_ERR | Interner Fehler. |
| CI_RET_START_FN_DB_TOO_LONG | Pfad Datenbasis zu lang. |
| CI_RET_START_FN_FW_TOO_LONG | Pfad Firmware zu lang. |
| CI_RET_START_FW_INIT_EXCP | Interner Fehler -> Firmware meldet Exception. |
| CI_RET_START_FW_INIT_TIMEOUT | Interner Fehler -> Firmware antwortet nicht. |
| CI_RET_START_INVALID_CP_NAME | Ungültiger CP Name. |
| CI_RET_START_INVALID_MODE | Ungültiger Start Mode. |
| CI_RET_START_LEN_DATABASE | Datenbasis File ist zu lang. |

| **Klasse DP_ERROR_CI**<br>**Fehlercodes** | **Bedeutung** |
|---|---|
| CI_RET_START_LEN_FIRMWARE | Firmware File ist zu lang |
| CI_RET_START_MAP | Interner Fehler. |
| CI_RET_START_MAX_CP | Maximale Anzahl von CPs schon erreicht. |
| CI_RET_START_NO_VIEW | Interner Fehler. |
| CI_RET_START_OK_ERR | Interner Fehler. |
| CI_RET_START_OPEN_DATABASE | Datenbasis File konnte nicht geöffnet werden. |
| CI_RET_START_OPEN_FIRMWARE | Firmware File konnte nicht geöffnet werden. |
| CI_RET_START_REG_DATABASE | Interner Fehler keine Datenbasis eingetragen. |
| CI_RET_START_REG_DOWNLOAD | Interner Fehler keine Firmware eingetragen. |
| CI_RET_START_REG_gc_data_0<br>CI_RET_START_REG_gc_data_1<br>CI_RET_START_REG_gc_group<br>CI_RET_START_REG_tmsi<br>CI_RET_START_REG_tmsi_reserve<br>CI_RET_START_REG_ttr_div_256<br>CI_RET_START_REG_tth_div_256_equ_dis | Eintrag fehlt in der Registry -> Fehler bei der Installation. |
| CI_RET_START_REG_NO_SWITCH_OUTPUT | Fehlerhafter Registry-Eintrag. |
| CI_RET_START_TEMP_LOCKED | Dienst momentan nicht möglich. |
| CI_RET_START_TRANSLATE | Interner Fehler. |

| **Klasse DP_ERROR_CI**<br>**Fehlercodes** | **Bedeutung** |
|---|---|
| CI_RET_CmResourceTypeDefault<br>CI_RET_CmResourceTypeDeviceSpecific<br>CI_RET_CmResourceTypeDma<br>CI_RET_CmResourceTypePort<br>CI_RET_HalAssignSlotResources<br>CI_RET_InterruptMode<br>CI_RET_IoReportResourceUsage<br>CI_RET_MAP_00<br>CI_RET_MAP_01<br>CI_RET_MAP_02<br>CI_RET_MAP_03<br>CI_RET_MAP_DOWN_0<br>CI_RET_MAP_DOWN_1<br>CI_RET_MAP_DPR_0<br>CI_RET_MAP_DPR_1<br>CI_RET_MAP_PLX_0<br>CI_RET_MAP_PLX_1<br>CI_RET_mem_def<br>CI_RET_mem_res_count<br>CI_RET_plx_length<br>CI_RET_UNMAP | Interne Fehler beim Identifizieren des CP über das Plug-and-Play-Bios.<br>Abhilfe -> Kontakt CP/Rechner überprüfen \| CP tauschen \| Rechner tauschen. |

## Klasse DP_ERROR_RES

| Klasse DP_ERROR_RES<br>Fehlercodes | Bedeutung |
|---|---|
| DP_RET_CP_MEMORY | Interner Speicherfehler |
| DP_RET_CP_MEMORY_1 | Interner Speicherfehler(1) |
| DP_RET_CP_MEMORY_2 | Interner Speicherfehler(2) |
| DP_RET_CP_MEMORY_3 | Interner Speicherfehler(3) |
| DP_RET_CP_MEMORY_4 | Interner Speicherfehler(4) |
| DP_RET_CP_MEMORY_5 | Interner Speicherfehler(5) |
| DP_RET_CP_MEMORY_6 | Interner Speicherfehler(6) |
| DP_RET_CP_MEMORY_7 | Interner Speicherfehler(7) |
| DP_RET_CP_NO_BUS_PAR | Es sind keine Busparameter in der Datenbasis des CP enthalten. |
| DP_RET_CP_NO_DP_PAR | Es sind keine DP-Parameter in der Datenbasis des CP enthalten. |
| DP_RET_CP_TOO_MANY_SLV | Es sind zu viele Slaves projektiert. |
| DP_RET_CP_TOO_MANY_USR | Es können sich keine weiteren DP-Instanzen beim CP anmelden. |
| DP_RET_MEMORY | Interner Speicherfehler |
| DP_RET_TOO_MANY_USR | Es können sich keine weiteren DP-Instanzen beim CP anmelden. |

| Klasse DP_ERROR_RES<br>Fehlercodes für die Slave-Modulfunktionen des CP 5614 | Bedeutung |
|---|---|
| DPS_RET_LESS_MEM | Die angeforderten Puffergrößen sind zu groß. |
| DPS_RET_NO_DPR_PTR | Kein Zugriff auf das DP-RAM. |
| DPS_RET_NO_SLAVE_MODULE | Slave-Modul nicht vorhanden. |

## 4.5 Formate der Slave-Daten

### Reihenfolge der Slave-Daten

Die Reihenfolge der Daten entspricht den konfigurierten Ein-/Ausgabe-Ports der DP-Slaves.

Beispielsweise werden die Eingabe-Ports einer ET 200B-16DI-Station wie folgt abgelegt: Port 0 im ersten Byte, Port 1 im 2. Byte, usw.

### Format von Datenwörtern

Bei der Ablage von Werten im Wortformat (2-Byte-Zahlen) ist die nachstehende Reihenfolge vorgeschrieben:

Zuerst wird das High-Byte (niederwertige Adresse), danach das Low-Byte (höherwertige Adresse) des Wortes eingetragen (d. h. Motorola-Format).

---

**Hinweis 1**

Diese Reihenfolge entspricht nicht dem Format der Prozessoren der 80x86/Pentium-Familie!

---

**Hinweis 2**

Beachten Sie bitte eventuelle Zusatzinformationen der Hersteller zu den Slaves, die Ihr Anwenderprogramm unterstützen soll.

---

## 4.6 Formate der Slave-Diagnosedaten

### Erklärung

In den nachfolgenden Datenstrukturen sind teilweise einzelne Bits in den Bytes von Bedeutung. Die Nummerierung der Bits ist dabei stets wie folgt: das niederwertigste Bit (LSB - **L**east **S**ignificant **B**it) hat die Nummer 0 und das höchstwertige (MSB - **M**ost **S**ignificant **B**it) die Nummer 7.

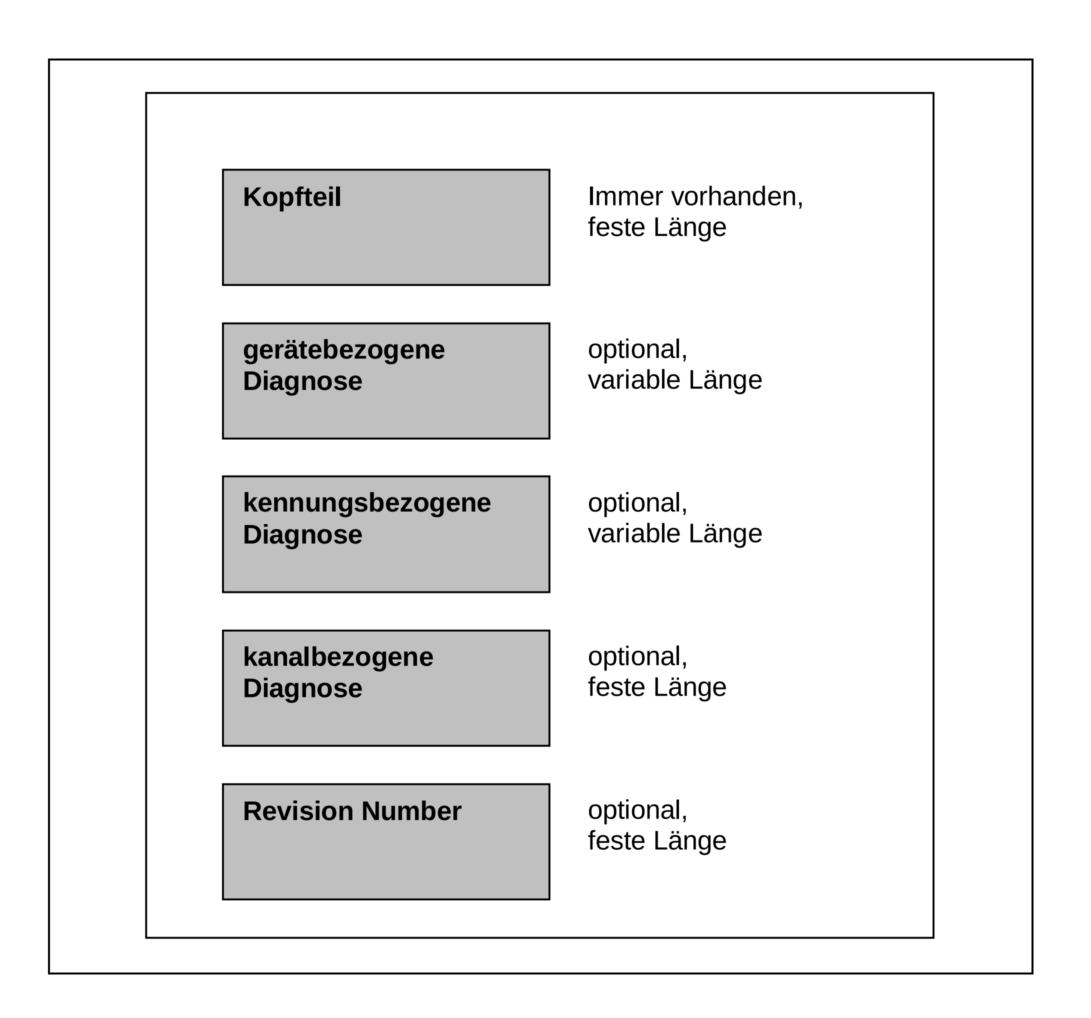
### 4.6.1 Übersicht der Gesamtstruktur

#### Mögliche Länge

Die Länge der Diagnosedaten liegt typischerweise im Bereich von 6 bis 32 Byte. Die maximal mögliche Länge beträgt 244 Byte.

#### Kopfteil und bis zu vier Folgeblöcke

Diagnosedaten haben einen festen Kopfteil und bis zu vier weitere Blöcke:




#### Identifikation der optionalen Teile

Die optionalen Teile enthalten an ihren Anfängen jeweils eindeutige Codierungen, an denen sie voneinander unterscheidbar sind.

### 4.6.2 Format des Kopfteils der Diagnose

#### Gesamtstruktur des Kopfteils

| Byte | Bedeutung |
|---|---|
| Byte 1 | Stationsstatus_1 |
| Byte 2 | Stationsstatus_2 |
| Byte 3 | Stationsstatus_3 |
| Byte 4 | Diag.Master_Add |
| Byte 5 und 6 | Ident_Number |

## Byte 1 (Stationsstatus_1)

Jedes Bit im Byte 1, dem „Stationstatus_1“-Byte, hat eine besondere Bedeutung.

| Bit | Name und Bedeutung |
|---|---|
| 7 | **Diag. Master_Lock** - Der DP-Slave ist bereits von einem anderen Master parametriert worden, d. h. der eigene Master hat momentan keinen Zugriff auf diesen Slave. |
| 6 | **Diag. Prm_Fault** - Dieses Bit wird vom DP-Slave gesetzt, falls das letzte Parametriertelegramm fehlerhaft war (beispielsweise falsche Länge, falsche Ident-Number, ungültige Parameter). |
| 5 | **Diag. Invalid_Slave_Response** - Dieses Bit wird gesetzt, sobald von einem angesprochenen DP-Slave eine unplausible Antwort empfangen wird. |
| 4 | **Diag. Not_Supported** - Dieses Bit wird gesetzt, sobald eine Funktion angefordert wurde, die von diesem Slave nicht unterstützt wird (beispielsweise wird ein Betrieb im SYNC-Modus gefordert, vom Slave aber nicht unterstützt). |
| 3 | **Diag. Ext_Diag** - Dieses Bit wird vom DP-Slave gesetzt. Ist das Bit gesetzt, so muss in dem Slave-spezifischen Diagnosebereich (Ext_Diag_Data) ein Diagnoseeintrag vorliegen. Ist das Bit nicht gesetzt, so kann in dem Slave-spezifischen Diagnosebereich (Ext_Diag_Data) eine Statusmeldung vorliegen. Die Bedeutung dieser Statusmeldung ist anwenderprogrammspezifisch zu vereinbaren. |
| 2 | **Diag. Cfg_Fault** - Dieses Bit wird gesetzt, sobald die vom DP-Master zuletzt gesendeten Konfigurationsdaten mit denjenigen, die der DP-Slave ermittelt hat, nicht übereinstimmen, d. h. es liegt ein Konfigurationsfehler vor. |
| 1 | **Diag. Station_Not_Ready** - Dieses Bit wird gesetzt, wenn der DP-Slave noch nicht für den Produktivdatenaustausch bereit ist. |
| 0 | **Diag. Station_Non_Existent** - Dieses Bit setzt der DP-Master, falls der DP-Slave nicht über den Bus erreichbar ist. Ist dieses Bit gesetzt, so enthalten die Diagnosebits den Zustand der letzten Diagnosemeldung oder den Initialwert. Der DP-Slave setzt dieses Bit fest auf Null. |

## Byte 2 (Stationsstatus_2)

Jedes Bit im Byte 2, dem „Stationsstatus_2“-Byte, hat eine besondere Bedeutung.

| Bit | Bedeutung |
|---|---|
| 7 | **Diag. Deactivated,** dieses Bit wird gesetzt, sobald der DP-Slave im lokalen Parametersatz als nicht aktiv gekennzeichnet und aus der zyklischen Bearbeitung herausgenommen wurde. |
| 6 | reserviert |
| 5 | **Diag. Sync_Mode,** dieses Bit wird vom DP-Slave gesetzt, sobald dieser DP-Slave das Sync-Steuerkommando erhalten hat. |
| 4 | **Diag. Freeze_Mode,** dieses Bit wird vom DP-Slave gesetzt, sobald dieser DP-Slave das Freeze-Steuerkommando erhalten hat. |
| 3 | **Diag. WD_On (Watchdog on),** dieses Bit wird vom DP-Slave gesetzt. Ist dieses Bit auf 1 gesetzt, so ist die Ansprechüberwachung beim DP-Slave aktiviert. |
| 2 | Dieses Bit wird vom DP-Slave fest auf 1 gesetzt. |
| 1 | **Diag. Stat_Diag (Statische Diagnose),** setzt der DP-Slave dieses Bit, so muss der DP-Master solange Diagnoseinformationen abholen, bis dieses Bit wieder gelöscht wird. Der DP-Slave setzt zum Beispiel dieses Bit, wenn er keine gültigen Nutzdaten zur Verfügung stellen kann. |
| 0 | **Diag. Prm_Req,** setzt der DP-Slave dieses Bit, so muss er neu parametriert und konfiguriert werden. Das Bit bleibt solange gesetzt, bis eine Parametrierung erfolgt ist (sind Bit 1 und Bit 0 gesetzt, so hat Bit 0 die höhere Priorität). |

## Byte 3 (Stationsstatus_3)

| Bit | Bedeutung |
|---|---|
| 7 | **Diag. Ext_Diag_Overflow,** ist dieses Bit gesetzt, so liegen mehr Diagnoseinformationen vor, als angegeben sind. Zum Beispiel setzt der DP-Slave dieses Bit, wenn mehr Kanaldiagnosen vorliegen, als der DP-Slave in seinen Sendepuffer eintragen kann; oder der DP-Master setzt dieses Bit, wenn der DP-Slave mehr Diagnosedaten sendet, als der DP-Master in seinem Diagnosepuffer berücksichtigen kann. |
| 6 bis 0 | reserviert |

## Byte 4 (Diag. Master_Add )

Hier wird die Adresse des DP-Masters eingetragen, der diesen DP-Slave parametriert hat. Ist der DP-Slave von keinem DP-Master parametriert, so setzt der DP-Slave in dieses Byte die Adresse 255 ein.

## Byte 5 und 6 (Ident_Number)

In die Bytes 5 und 6, den „Ident_Number“-Bytes, wird die Herstellerkennung für einen DP-Slave-Typ vergeben. Diese Kennung kann zum einen für Prüfungszwecke und zum anderen zur genauen Identifizierung herangezogen werden.
Der Wert liegt im Motorola-Format vor, d. h. der höherwertige Anteil ist im Byte 5.

### 4.6.3 Format der gerätebezogenen Diagnose (Standard-DP-Slave)

#### Bedeutung

Bei Standart-DP-Slaves (ohne DPC1-Erweiterungen) werden in diesem Block allgemeine Diagnoseinformationen wie Beispiel Übertemperatur, Unterspannung oder Überspannung hinterlegt. Die Kodierung wird gerätespezifisch durch die Slave-Hersteller festgelegt. Zur weiteren Auswertung muss die Ident_Number des Slave herangezogen werden.

#### Struktur

| Byte | Bedeutung |
|---|---|
| Byte 1 | Header-Byte |
| Byte 2 bis 63 | Diagnostic_User_Data |

#### Byte 1 (Header-Byte)

Die beiden höchstwertigen Bits im ersten Byte haben den Wert 00. Damit wird der Block „Gerätebezogene Diagnose“ als ganzes identifiziert.

Die restlichen sechs Bits im ersten Byte geben die Länge des Datenblocks inklusive dem ersten Byte an.

### 4.6.4 Format der gerätebezogenen Diagnose (Slaves mit DP-V1-Erweiterungen)

#### Bedeutung

Bei diesen Slaves enthält die gerätebezogene Diagnose Alarmdaten oder Statusmeldungen. Bei den Statusmeldungen gibt es zusätzlich die Variante „Modulstatus“.

#### Gesamtstruktur mit zwei Varianten

Die gerätebezogene Diagnose gibt es in zwei Kodiervarianten, nämlich Alarme und Statusmeldungen. Einige Komponenten haben entsprechend unterschiedliche Kodierungen.

Die beiden Varianten können am Byte 2, Bit 7 auseinandergehalten werden.

| Byte | Variante „Alarme“ | Variante „Statusmeldungen“ |
|---|---|---|
| Byte 1 | Header-Byte | |
| Byte 2 | Alarm_Type | Status_Type |
| Byte 3 | Slot-Nummer | |
| Byte 4 | Alarm-Specifier | Status-Specifier |
| Byte 5 bis 63 | Diagnostic_User_Data | |

#### Byte 1 (Header-Byte)

Die beiden höchstwertigen Bits im ersten Byte haben den Wert 00. Damit wird der Block „Gerätebezogene Diagnose“ als ganzes identifiziert.

Die restlichen sechs Bits im ersten Byte geben die Länge des Datenblocks inklusive dem ersten Byte an.

### Byte 2 (Variante Alarm_Type)

<table>
<tr><th>Bit</th><th colspan="2">Bedeutung</th></tr>
<tr><td rowspan="2">7</td><td>Wert</td><td>Bedeutung</td></tr>
<tr><td>0</td><td>Alarm</td></tr>
<tr><td>6 bis 0</td><td colspan="2">Alarm_Type<br>0 = reserviert<br>1 = Diagnose Alarm<br>2 = Prozess Alarm<br>3 = Pull Alarm<br>4 = Plug Alarm<br>5 = Status Alarm<br>6 = Update Alarm<br>7-31 = reserviert<br>32-126 = herstellerspezifisch<br>127 = reserviert</td></tr>
</table>

### Byte 2 (Variante Status_Type)

<table>
<tr><th>Bit</th><th colspan="2">Bedeutung</th></tr>
<tr><td rowspan="2">7</td><td>Wert</td><td>Bedeutung</td></tr>
<tr><td>1</td><td>Statusmeldung</td></tr>
<tr><td>6 bis 0</td><td colspan="2">Status_Type<br>0 = reserviert<br>1 = Status Meldung<br>2 = Modul_Status<br>3-31 = reserviert<br>32-126 = herstellerspezifisch<br>127 = reserviert</td></tr>
</table>

### Byte 3 (Variante Slot-Nummer)

| Bit | Bedeutung |
|---|---|
| 7 bis 0 | Slot-Nummer |

### Byte 4 (Variante Alarm-Specifier)

| Bit | Bedeutung | |
|---|---|---|
| 7 bis 3 | Seq_Nr | Eindeutige Identifikation einer Alarm Meldung |
| 2 | Add_Ack | Ist dieses Bit gesetzt, so zeigt der DP-V1 Master an, dass dieser Alarm zusätzlich zur DPC1 Alarm Acknowledge eine separate User-Bestätigung (z. B. in Form eines Write Dienstes) erfordert. |
| 1 bis 0 | Alarm Specifier | 0 = **keine weitere Unterscheidung**<br>1 = **Alarm erscheint, Slot gestört** Der Slot generiert einen Alarm wegen eines Fehlers.<br>2 = **Alarm verschwindet, Slot ok** Der Slot generiert einen Alarm und zeigt an, dass der Slot keine weiteren Fehler hat.<br>3 = **Alarm verschwindet, Slot weiterhin gestört** Der Slot generiert einen Alarm und zeigt an, dass der Slot weiterhin Fehler hat. |

### Byte 4 (Variante Status-Specifier)

| Bit | Bedeutung |
|---|---|
| 7 bis 2 | reserviert |
| 1 bis 0 | Status Specifier<br>0 = keine weitere Unterscheidung<br>1 = Status erscheint<br>2 = Status verschwindet<br>3 = reserviert |

## Byte 5-63

### User-spezifische Information

Diese Bytes enthalten Daten mit zusätzlicher user-spezifischer Information. Das Format wird in der Slave-Dokumentation beschrieben.

Bei der Variante „Statusmeldungen“ **und** der Einstellung „Modul_Status“ (siehe Byte 2) jedoch sind für jedes Modul/Slot zwei Bits vorgesehen. Der Modul_Status ist Byte-orientiert, nicht verwendete Bits werden auf 0 gesetzt.

| | **MSB** | | | | | | | **LSB** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| **Byte 5:** | Modul_ Status 4 | | Modul_ Status 3 | | Modul_ Status 2 | | Modul_ Status 1 | |
| **...** | ... | | ... | | ... | | ... | |
| **Byte m:** | Modul_ Status n | | Modul_ Status n-1 | | ... | | ... | |

Die Status-Bits sind jeweils wie folgt kodiert:

| Bit | Bedeutung |
| --- | --- |
| 00 | Daten gültig |
| 01 | Daten ungültig: Die Daten des zugehörigen Moduls sind ungültig wegen eines Fehlers (zum Beispiel Kurzschluss) |
| 10 | Daten ungültig/falsches Modul: Die Daten des zugehörigen Moduls sind ungültig wegen eines falschen Moduls |
| 11 | Daten ungültig/ kein Modul: Die Daten des zugehörigen Moduls sind ungültig weil kein Modul gesteckt ist |

## Beispiel einer gerätebezogenen Diagnose mit Statusmeldung

Im folgenden Bild ist eine Statusdiagnose nach dem oben aufgeführten Schema dargestellt.

| | MSB | | | | | | | LSB |
|---|---|---|---|---|---|---|---|---|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **Gerätebezogene Diagnose:** | **0** | **0** | 0 | 0 | 0 | 1 | 1 | 1 |
| Statustyp: Statusmeldung | **1** | 0 | 0 | 0 | 0 | 0 | 0 | **1** |
| Slot-Nummer 2 | 0 | 0 | 0 | 0 | 0 | 0 | **1** | 0 |
| Specifier: keine weitere Unterscheidung | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| User-Diagnosedaten: 5 Durchschnittstemperatur | 0 | 0 | 0 | 0 | 0 | **1** | 0 | **1** |
| User-Diagnosedaten: Temperaturwert (Unsigned 16) | x | x | x | x | x | x | x | x |
| | x | x | x | x | x | x | x | x |

## Beispiel einer gerätebezogenen Diagnose mit Alarmmeldung

Im folgenden Bild ist eine Alarmdiagnose nach dem oben aufgeführten Schema dargestellt.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Gerätebezogene Diagnose:** | **0** | **0** | 0 | 0 | 1 | 0 | 0 | 1 |
| Alarmtyp: Prozessalarm | **0** | 0 | 0 | 0 | 0 | 0 | **1** | 0 |
| Slot-Nummer 3 (Ventil B) | 0 | 0 | 0 | 0 | 0 | 0 | **1** | **1** |
| Specifier: Alarm erscheint | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **1** |
| User-Diagnosedaten: 0x50 (oberer Grenzdruck überschritten) | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| User-Diagnosedaten (Time of day, 4 Byte) | x | x | x | x | x | x | x | x |
| | x | x | x | x | x | x | x | x |
| | x | x | x | x | x | x | x | x |
| | x | x | x | x | x | x | x | x |

### 4.6.5 Format der kennungsbezogenen Diagnose

#### Bedeutung

Bei modularen Slaves mit je einem Kennungsbyte pro Modul ist eine modulspezifische Diagnose möglich. Der DP-Master sendet die Kennungsbytes während der Anlaufphase in einem Konfiguriertelegramm an den Slave. In der kennungsbezogenen Diagnose ist jedem Modul im Datenblock ein Bit zugeordnet. Ein gesetztes Bit bedeutet, dass in dem zugehörigen Modul eine Diagnose vorliegt.

#### Identifikation der kennungsbezogenen Diagnose und Länge im ersten Byte

Die beiden höchstwertigen Bits im ersten Byte haben den Wert 01.

Die restlichen sechs Bits im ersten Byte geben die Länge des Datenblocks inklusive dem ersten Byte an.

#### Inhalt des Datenblocks

Die restlichen Bytes des Datenblocks enthalten einen Bitarray. Das niederwertigste Bit des 2. Bytes hat den Index 0, und so weiter aufsteigend, so dass z. B. das höchstwertige Bit des 3. Bytes den Index 15 hat.

Die Bits geben jeweils an, ob für die Kennungsindizes Diagnosen gemeldet wurden.

#### Bedeutung der Bits in den Diagnosebytes

Jedes Bit in den „Diagnosebytes“ hat eine besondere Bedeutung.

---

**Hinweis**

Beachten Sie das Beispiel am Ende des Kapitels 4.6.6.

---

### 4.6.6 Format der kanalbezogenen Diagnose

#### Bedeutung

Für die einzelnen Kanäle eines Slave wird hier die Datenorganisation des Kanals sowie Meldungen, wie z. B. „Unterspannung“ oder „Kurzschluss“, geliefert.

---

**Hinweis**

Beachten Sie das Beispiel weiter unten.

---

#### Folge von Einträgen zu je drei Byte

Die kanalbezogene Diagnose besteht aus einer Folge von Einträgen, die alle dasselbe Format haben und je drei Byte lang sind (Header-Byte, Kanalnummer, Art der Diagnose).

#### Header-Byte (Kennungsnummer) mit Identifikation

Die beiden höchstwertigen Bits im ersten Byte jedes Eintrags haben den Wert 10.

Die restlichen sechs Bits im ersten Byte geben eine Kennungsnummer an. Für diese Kennungsnummer ist typischerweise in der kennungsspezifischen Diagnose das entsprechende Bit gesetzt.

## Kanalnummer

<table>
<tr><th>Bit</th><th>Bedeutung</th></tr>
<tr><td>7 und 6</td><td>Ein-/Ausgabe
<table>
<tr><th>Wert</th><th>Bedeutung</th></tr>
<tr><td>00</td><td>reserviert</td></tr>
<tr><td>01</td><td>Eingabe</td></tr>
<tr><td>10</td><td>Ausgabe</td></tr>
<tr><td>11</td><td>Eingabe/Ausgabe</td></tr>
</table>
</td></tr>
<tr><td>5 bis 0</td><td>Kanalnummer 0 bis 63</td></tr>
</table>

**Hinweis**

Bei Kennungsbytes die sowohl Ein- als auch Ausgaben enthalten, wird in Bit 7 und Bit 6 der Kanalnummer die Richtung des diagnostizierten Kanals angezeigt.

## Art der Diagnose

<table>
<tr><th>Bit</th><th>Bedeutung</th></tr>
<tr><td>7 bis 5</td><td>Kanaltyp
<table>
<tr><th>Kanaltyp</th><th>Bedeutung</th></tr>
<tr><td>000</td><td>reserviert</td></tr>
<tr><td>001</td><td>Bit</td></tr>
<tr><td>010</td><td>2 Bit</td></tr>
<tr><td>011</td><td>4 Bit</td></tr>
<tr><td>100</td><td>Byte</td></tr>
<tr><td>101</td><td>Wort</td></tr>
<tr><td>110</td><td>2 Worte</td></tr>
<tr><td>111</td><td>reserviert</td></tr>
</table>
</td></tr>
<tr><td>4 bis 0</td><td>Fehlertyp
<table>
<tr><th>Fehlertyp</th><th>Bedeutung</th></tr>
<tr><td>0</td><td>reserviert</td></tr>
<tr><td>1</td><td>Kurzschluss</td></tr>
<tr><td>2</td><td>Unterspannung</td></tr>
<tr><td>3</td><td>Überspannung</td></tr>
<tr><td>4</td><td>Überlast</td></tr>
<tr><td>5</td><td>Übertemperatur</td></tr>
<tr><td>6</td><td>Leitungsbruch</td></tr>
<tr><td>7</td><td>Oberer Grenzwert überschritten</td></tr>
<tr><td>8</td><td>Unterer Grenzwert unterschritten</td></tr>
<tr><td>9</td><td>Fehler</td></tr>
<tr><td>10-15</td><td>reserviert</td></tr>
<tr><td>16-31</td><td>herstellerspezifisch</td></tr>
</table>
</td></tr>
</table>

## 4.6.7 Format der Revision Number

### Bedeutung

Es wird die Revision Number des Slaves eingetragen. Der Eintrag besteht aus einem einzelnen Byte..

### Struktur

| Byte | Bedeutung |
|---|---|
| Byte 1 | Revision Number |

### Byte 1

Die beiden höchstwertigen Bits im ersten Byte haben den Wert 11. Damit wird der Block „Revision Number“ als ganzes identifiziert.

Die restlichen sechs Bits enthalten die Revision Number des Slave.

## Beispiel einer Diagnose nach obigem Schema

Im folgenden Bild ist eine Diagnose nach dem oben aufgeführten Schema dargestellt.

| | MSB | | | | | | | LSB |
|---|---|---|---|---|---|---|---|---|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **Kopfteil** | 6 Byte lang | | | | | | | |
| **Gerätebezogene Diagnose:** (Header-Byte) | **0** | **0** | 0 | 0 | 0 | 1 | 0 | 0 |
| Gerätespezifisches Diagnosefeld | Die Bedeutung der Bits ist herstellerspezifisch, hier 3 Byte lang | | | | | | | |
| **Kennungsbezogene Diagnose:** (Header-Byte) | **0** | **1** | 0 | 0 | 0 | 1 | 0 | 1 |
| Kennungsnummer **0** mit Diagnose (Diagnosebyte) | 0 7. | 0 6. | 0 5. | 0 4. | 0 3. | 0 2. | 0 1. | **1** 0. |
| Kennungsnummer **12** mit Diagnose (Diagnosebyte) | 0 15. | 0 14. | 0 13. | **1** 12. | 0 11. | 0 10. | 0 9. | 0 8. |
| Kennungsnummer **18** mit Diagnose (Diagnosebyte) | 0 23. | 0 22. | 0 21. | 0 20. | 0 19. | **1** 18. | 0 17. | 0 16. |
| (Diagnosebyte) | 0 31. | 0 30. | 0 29. | 0 28. | 0 27. | 0 26. | 0 25. | 0 24. |
| **Kanalbezogene Diagnose:** Kennungsnummer **0** (Header-Byte) | **1** | **0** | 0 | 0 | 0 | 0 | 0 | 0 |
| Kanal 2 (Kanalnummer) | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| Überlast, Kanal bitweise organisiert (Art der Diagnose) | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| Kennungsnummer **12** (Header-Byte) | **1** | **0** | 0 | 0 | 1 | 1 | 0 | 0 |
| Kanal 6 (Kanalnummer) | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| Oberer Grenzwert überschritten, Kanal wortweise organisiert (Art der Diagnose) | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |

## 4.7 Format der Slave-Parameterdaten

### Einordnung

Nachfolgend werden die Formate der Slave-Parameterdaten beschrieben, wie sie vom Aufruf DP_read_slv_par im Kapitel 4.1.11 erfragt werden. Es sind drei Varianten möglich, die beim Aufruf von DP_read_slv_par durch den Wert des Parameters „type“ ausgewählt werden:

| Variante | Wert des Parameters „type“ |
|---|---|
| Allgemeine Slave-Parameter | DP_SLV_TYP |
| Parametrierdaten | DP_SLV_PRM |
| Konfigurierdaten | DP_SLV_CFG |

Nachfolgend wird der Aufbau der drei Varianten näher erläutert.

## 4.7.1 Aufbau der allgemeinen Slave-Parameter

### Übersicht

Es handelt sich jeweils um Bytes. Die Komponenten sind byte-aligned, d. h. sie stehen unmittelbar hintereinander. Die grau hinterlegten Parameter sind nur für DP-Slaves mit DP-V1-Zusatzfunktionen relevant (werden zur Zeit nur teilweise unterstützt). Soweit notwendig werden die Parameter weiter unten näher beschrieben.

| Name | Bedeutung (s. u.) |
|---|---|
| Sl_Flag | Aktiviert, Fail-Safe-Verhalten u. a. |
| Slave_Typ | Herstellerspezifisch |
| Max_Diag_Data_Len | Max. Länge Diagnosedaten |
| Max_Alarm_Len | Max. Länge Alarmdaten |
| Max_Channel_Data_Len | Max. Länge DPC1-Telegramm |
| Diag_Upd_Delay | Diagnosezähler |
| Alarm_Mode | Max. Anzahl paralleler Alarme |
| Add_Sl_Flag | Daten zu AUTOCLEAR u. a. |
| Byte 9 bis 14 | reserviert |

### Byte-Aufbau

In den nachfolgenden Datenstrukturen sind teilweise einzelne Bits in den Bytes von Bedeutung. Die Nummerierung der Bits ist dabei stets wie folgt: das niederwertigste Bit (LSB - **L**east **S**ignificant **B**it) hat die Nummer 0 und das höchstwertige (MSB - **M**ost **S**ignificant **B**it) die Nummer 7.

## Sl_Flag

Dieser Parameter enthält Slave-bezogene Flags.

Jedes Bit im „Sl_Flag"-Byte hat eine besondere Bedeutung.

| Bit | Bedeutung |
|---|---|
| 7 | **Active**<br>0 bedeutet: DP-Slave ist nicht aktiviert<br>1 bedeutet: DP-Slave ist aktiviert |
| 6 | **New_Prm**<br>0 bedeutet: DP-Slave bekommt Nutzdaten<br>1 bedeutet: DP-Slave bekommt neue Parametrierdaten |
| 5 | **Fail_Safe**<br>0 bedeutet: DP-Slave bekommt Bytes mit Dateninhalt 0 im CLEAR-Modus<br>1 bedeutet: DP-Slave bekommt Leertelegramme im CLEAR-Modus |
| 4 | reserviert |
| 3 | **DPV1_Supported**<br>0 bedeutet: DP-Slave-Funktionalität nach EN50170<br>1 bedeutet: DP-Slave-Funktionalität nach DP-V1 |
| 2 | **DP-V1-Daten-Typen**<br>0 bedeutet: DP-Slave Konfigurierdaten nach EN 50170<br>1 bedeutet: DP-Slave Konfigurierdaten nach DP-V1 |
| 1 | **Extra Alarm SAP**<br>0 bedeutet: DP-Master quittiert Alarme via SAP 51<br>1 bedeutet: DP-Master quittiert Alarme via SAP 50 |
| 0 | reserviert |

## Slave-Typ

Dieser Parameter enthält eine herstellerspezifische Typbezeichnung für das Slave-Gerät.

| Wert | Bedeutung |
|---|---|
| 0 | DP-Norm-Slave |
| 1 bis 15 | reserviert |
| 16 bis 255 | herstellerspezifisch |

## Alarm_Mode

Dieser Parameter legt die maximale Anzahl möglicher aktiver Alarme fest.

| Wert | Bedeutung |
|---|---|
| 0 | 1 Alarm von jedem Alarmtyp |
| 1 | 2 Alarme insgesamt |
| 2 | 4 Alarme insgesamt |
| 3 | 8 Alarme insgesamt |
| 4 | 12 Alarme insgesamt |
| 5 | 16 Alarme insgesamt |
| 6 | 24 Alarme insgesamt |
| 7 | 32 Alarme insgesamt |

## Add_Sl_Flag

| Bit | Name und Bedeutung |
|---|---|
| 7-2 | reserviert |
| 1 | **Ignore_AClr**<br>0 bedeutet: AUTOCLEAR ausführen<br>1 bedeutet: AUTOCLEAR ignorieren |
| 0 | **NA_To_Abort**<br>0 bedeutet: Slave wird nicht initialisiert, wenn er nicht antwortet.<br>1 bedeutet: Slave wird neu initialisiert, wenn er nicht antwortet. |

## 4.7.2 Aufbau der Parametrierdaten

### Übersicht

Die Parametrierdaten setzen sich aus busspezifischen Daten und DP-Slave spezifischen Daten zusammen.

| | Byte-Nummer | Bedeutung |
|---|---|---|
| Kopfdaten | Byte 1 | Station_status |
| | Byte 2 | WD_Fact_1 |
| | Byte 3 | WD_Fact_2 |
| | Byte 4 | Min. Station Delay Responder |
| | Byte 5-6 | Ident_Number |
| | Byte 7 | Group_Ident |
| User-Parametrierdaten | Byte 8 | DPV1_Status_1 – herstellerspezifisch bei Slaves ohne DPV1-Funktionalität |
| | Byte 9 | DPV1_Status_2 – herstellerspezifisch bei Slaves ohne DPV1-Funktionalität |
| | Byte 10 | DPV1_Status_3 – herstellerspezifisch bei Slaves ohne DPV1-Funktionalität |
| | Byte 11-n | herstellerspezifische Daten |

### Byte 1 (Station_status)

Byte 1, der „Station_status“, hat folgenden Aufbau:

Dabei hat jedes Bit im „Station_status“-Byte eine besondere Bedeutung.

<table>
<tr><th>Bit</th><th>Bedeutung</th></tr>
<tr><td>7 und 6</td><td>Lock_Req und Unlock_Req<br>
<table>
<tr><th>Bit 7</th><th>Bit 6</th><th>Bedeutung</th></tr>
<tr><td>0</td><td>0</td><td>Bei einer Parametrierung wird die min $T_{SDR}$ überschrieben. Alle anderen Parameter bleiben unbeeinflusst.</td></tr>
<tr><td>0</td><td>1</td><td>Der DP-Slave wird für andere Master freigegeben.</td></tr>
<tr><td>1</td><td>0</td><td>Der DP-Slave wird für andere Master gesperrt, alle Parameter werden übernommen (Ausnahme: min $T_{SDR}$ = 0).</td></tr>
<tr><td>1</td><td>1</td><td>Der DP-Slave wird für andere Master freigegeben.</td></tr>
</table></td></tr>
<tr><td>5</td><td>Sync_Req<br>Mit diesem Bit wird dem Slave angezeigt, dass er im Sync-Modus betrieben werden soll, sobald das Kommando mit der Funktion DP_global_crtl() übergeben wird.</td></tr>
<tr><td>4</td><td>Freeze_Req<br>Mit diesem Bit wird dem DP-Slave angezeigt, dass er im Freeze-Modus betrieben werden soll, sobald das Kommando mit der Funktion DP_global_ctrl() übergeben wird.</td></tr>
<tr><td>3</td><td>Watchdog<br>Ist dieses Bit auf Null gesetzt, so ist die Ansprechüberwachung deaktiviert. Ist das Bit gesetzt, wird die Ansprechüberwachung beim DP-Slave aktiviert.</td></tr>
<tr><td>2 bis 0</td><td>reserviert</td></tr>
</table>

### Byte 2 und 3 (WD_Fact_1 und WD_Fact_2)

Diese beiden Bytes enthalten Faktoren für die Einstellung der Ansprechüberwachungszeit ($T_{WD}$).

Die Ansprechüberwachungszeit sorgt in einem DP-Slave dafür, dass nach dem Ausfall des DP-Masters nach Ablauf dieser Zeit die Ausgänge den sicheren Zustand einnehmen:

$T_{WD}$ [ms] = 10 * WD_Fact_1 * WD_Fact_2

### Byte 4 (Min. Station Delay Responder)

Das ist die Zeit, die der DP-Slave warten muss, bis er seine Antworttelegramme an den DP-Master zurücksenden darf; Einheit: Bit-Zeiten.

### Byte 5 und 6 (Ident_Number)

Diese Nummer wird vom Hersteller vergeben. Der DP-Slave akzeptiert nur Parametriertelegramme, bei denen die übertragene Ident_Number mit der eigenen Ident_Number übereinstimmt. Der Wert wird im Motorola-Format übertragen, d. h. die höherwertige Hälfte steht in Byte 5.

### Byte 7 (Group_Ident)

Mit diesem Parameter kann eine Gruppenbildung für die Funktion DP_global_ctrl() gebildet werden. Jedes Bit stellt eine Gruppe dar. Die Group_Ident wird nur bei gesetztem Lock_Req-Bit übernommen (siehe Byte 1).

### Byte 8 (DPV1_Status_1)

| Bit | Bedeutung |
|---|---|
| 7 | **DPV1_Enable**<br>Dieses Bit wird gesetzt, um die DP-V1-Funktionalität eines DP-V1-Slave zu aktivieren. Ist dieses Bit nicht gesetzt, so arbeitet der Slave sofern möglich im Kompatibilitätsmodus. |
| 6 | **Fail Safe**<br>Dieses Bit wird gesetzt, um anzuzeigen, dass der Slave im Fail-Safe-Modus arbeitet. |
| 5-3 | reserviert |
| 2 | **WD_Base_1ms**<br>Ist dieses Bit gesetzt, so wird die Watchdog-Überwachungszeit nach folgender Formel berechnet:<br>$T_{WD}$ [ms] = WD_Fact_1 * WD_Fact 2<br>Ist dieses Bit nicht gesetzt, wird die Watchdog-Überwachungszeit wie folgt berechnet:<br>$T_{WD}$ [ms] = 10 * WD_Fact_1 * WD_Fact 2 |
| 1 bis 0 | reserviert |

### Byte 9 (DPV1_Status_2)

| Bit | Bedeutung |
|---|---|
| 7 | **Enable_Pull_Plug_Alarm**<br>Dieses Bit wird gesetzt, um die Meldung des Alarm Typs „Pull_Plug_Alarm“ zu ermöglichen. |
| 6 | **Enable_Process_Alarm**<br>Dieses Bit wird gesetzt, um die Meldung des Alarm Typs „Process_Alarm“ zu ermöglichen. |
| 5 | **Enable_Diagnostic_Alarm**<br>Dieses Bit wird gesetzt, um die Meldung des Alarm Typs „Diagnostic_Alarm“ zu ermöglichen. |
| 4 | **Enable_Manufacturer_Specific_Alarm**<br>Dieses Bit wird gesetzt, um die Meldung aller herstellerspezifischer Alarme zu ermöglichen. |
| 3 | **Enable_Status_Alarm**<br>Dieses Bit wird gesetzt, um die Meldung des Alarm Typs „Status_Alarm“ zu ermöglichen. |
| 2 | **Enable_Update_Alarm**<br>Dieses Bit wird gesetzt, um die Meldung des Alarm Typs „Update_Alarm“ zu ermöglichen. |
| 1 | reserviert |
| 0 | **Check_Cfg_Mode**<br>Durch dieses Bit kann die Reaktion auf Empfang von Konfigurationsdaten beeinflusst werden. Ist dieses Bit 0, so erfolgt die Prüfung wie in EN 50170 beschrieben. Ist dieses Bit gesetzt, kann die Prüfung auf eine user-spezifische Art und Weise erfolgen. |

## Byte 10

**DPV1_Status_3**

<table>
<tr><th>Bit</th><th>Bedeutung</th></tr>
<tr><td>7-3</td><td>reserviert</td></tr>
<tr><td>2-0</td><td>Alarm Modus<br>
<table>
<tr><th>Bit 4</th><th>Bedeutung</th></tr>
<tr><td>0</td><td>1 Alarm von jedem Typ</td></tr>
<tr><td>1</td><td>2 Alarme insgesamt</td></tr>
<tr><td>2</td><td>4 Alarme insgesamt</td></tr>
<tr><td>3</td><td>8 Alarme insgesamt</td></tr>
<tr><td>4</td><td>12 Alarme insgesamt</td></tr>
<tr><td>5</td><td>16 Alarme insgesamt</td></tr>
<tr><td>6</td><td>24 Alarme insgesamt</td></tr>
<tr><td>7</td><td>32 Alarme insgesamt</td></tr>
</table>
</td></tr>
</table>

## Byte 11 bis n

**herstellerspezifische Daten**

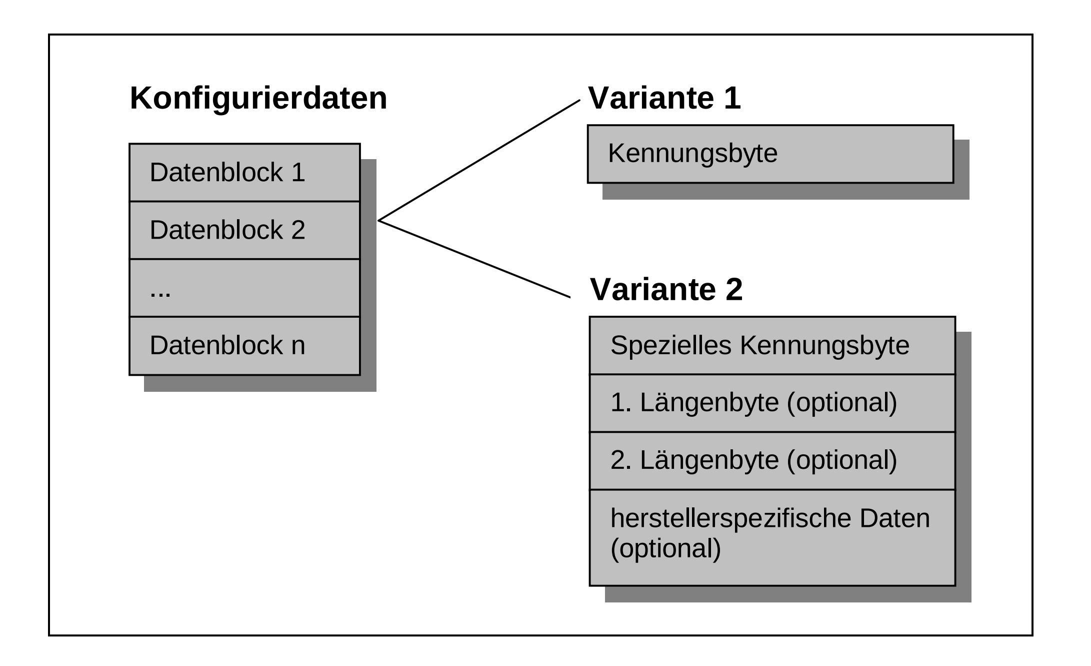
Diese Bytes sind für DP-Slave-spezifische Parameter (z. B. Diagnosefilter, Reglerparameter) frei belegbar. Die Bedeutung und die Wertebereiche werden herstellerspezifisch festgelegt.

### 4.7.3 Aufbau der Konfigurierdaten

#### Übersicht des Aufbaus

Die Konfigurierdaten enthalten den Umfang der Ein- und Ausgabedatenbereiche und Angaben über die Datenkonsistenz.

Sie bestehen aus einer Folge von Datenblöcken, wobei jeder Datenblock eines von zwei alternativen Formaten haben kann:




Die beiden Datenblockvarianten lassen sich an den Bits 4 und 5 des ersten Bytes auseinanderhalten.

#### Länge der Konfigurierdaten

Die Länge der Konfigurierdaten beträgt vorzugsweise 1 bis 32. Optional sind aber bis zu 244 Byte möglich.

## Aufbau eines Kennungsbytes

Jedes Bit hat eine besondere Bedeutung:

<table>
<tr><th>Bit</th><th>Bedeutung</th></tr>
<tr><td>0 bis 3</td><td>Anzahl der Dateneinheiten - 1<br>3 bedeutet z. B. 4 Dateneinheiten der in Bit 6 angegebenen Länge</td></tr>
<tr><td>4 und 5</td><td>Ein-/Ausgabe
<table>
<tr><th>Bit 5</th><th>Bit 4</th><th>Bedeutung</th></tr>
<tr><td>0</td><td>0</td><td>kommt nicht vor (s. u.)</td></tr>
<tr><td>0</td><td>1</td><td>Eingabe</td></tr>
<tr><td>1</td><td>0</td><td>Ausgabe</td></tr>
<tr><td>1</td><td>1</td><td>Ein-/Ausgabe</td></tr>
</table></td></tr>
<tr><td>6</td><td>Länge der Dateneinheiten<br>0 bedeutet: Die Dateneinheit ist ein Byte<br>1 bedeutet: Die Dateneinheit sind 2 Byte</td></tr>
<tr><td>7</td><td>Konsistenz<br>0 bedeutet: Byte- oder Wort-Konsistenz<br>1 bedeutet: Konsistenz über die gesamte Länge</td></tr>
</table>

## Aufbau eines speziellen Kennungsbytes

Spezielle Kennungsformate ermöglichen weitergehende Konfigurationen durch Erhöhung der Flexibilität. Jedes Bit im Kennungsbyte hat eine besondere Bedeutung.

<table>
<tr><th>Bit</th><th>Bedeutung</th></tr>
<tr><td>0 bis 3</td><td>Länge der herstellerspezifischen Daten<br>Diese Bits enthalten die Länge der herstellerspezifischen Daten.</td></tr>
<tr><td>4 und 5</td><td>Diese Bits sind fest mit 00 vorbelegt.</td></tr>
<tr><td>6 und 7</td><td>Ein-/Ausgabe
<table>
<tr><td>Bit 7</td><td>Bit 6</td><td>Bedeutung</td></tr>
<tr><td>0</td><td>0</td><td>Es folgen keine optionalen Bytes</td></tr>
<tr><td>0</td><td>1</td><td>Es folgt 1 Längen-Byte für Eingaben</td></tr>
<tr><td>1</td><td>0</td><td>Es folgt 1 Längen-Byte für Ausgaben</td></tr>
<tr><td>1</td><td>1</td><td>Es folgt:<br>• 1 Längen-Byte für Ausgaben<br>• 1 Längen-Byte für Eingaben</td></tr>
</table></td></tr>
</table>

## Längen-Bytes

Jedes Bit in den Längen-Bytes hat eine besondere Bedeutung.

| Bit | Bedeutung |
|---|---|
| 0 bis 5 | **Anzahl der Dateneinheiten**<br>0 (dezimal) bedeutet: 1 Dateneinheit<br>63 (dezimal) bedeutet: 64 Dateneinheiten<br>(zur Länge der Dateneinheiten siehe Bit 6) |
| 6 | **Länge**<br>0 bedeutet: eine Dateneinheit ist ein Byte lang<br>1 bedeutet: eine Dateneinheit ist zwei Bytes lang |
| 7 | **Konsistenz**<br>0 bedeutet: Konsistenz über Byte oder Wort<br>1 bedeutet: Konsistenz über die gesamte Länge des Moduls |

## Beispiel für spezielles Kennungsformat

Aus der nachfolgenden Tabelle entnehmen Sie ein Beispiel für ein spezielles Kennungsformat.

| Oktett | Bits im Byte | | | | | | | | Bedeutung |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | Aus-/Eingabe,<br>3 Byte herstellerspezifische Daten |
| 2 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | Konsistenz, Ausgabe, 16 Worte |
| 3 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | Konsistenz, Eingabe, 8 Worte |
| 4 bis 6 | | | | | | | | | herstellerspezifische Daten |

# FAQ (Frequently Asked Questions)

# 5

In diesem Kapitel werden, nach Kategorien geordnet, typische Detailfragen („Frequently Asked Questions“) zur Programmierschnittstelle des CP 5613 und CP 5614 beantwortet.

Für die Erstellung Ihres Anwenderprogramms sind die Abschnitte über die Strukturierung von Anwenderprogrammen und die Checkliste für Programmierer besonders wichtig.

Beachten Sie bitte auch die FAQ-Liste in der Installationsanleitung des Produkts.

## 5.1 FAQ zum Leistungsumfang des Produkts

### Welche Software-Versionen gibt es, und wie unterscheiden sie sich?

Beachten Sie dazu bitte die Versionstabelle im Kapitel 14.2 der Installationsanleitung.

### Welche Betriebssysteme werden unterstützt?

Beachten Sie dazu bitte die Versionstabelle im Kapitel 14.2 der Installationsanleitung.

### Welche Compiler und Programmiersprachen werden unterstützt?

Beachten Sie dazu bitte die Versionstabelle im Kapitel 14.2 der Installationsanleitung.

Für Anbindungen an Borland C++ und Delphi informieren Sie sich bitte bei der Firma AIXO (Internet: http://www.aixo.com).
Durch die Auslegung der „dp_base.dll“ mit Standard-Calling-Conventions ist die Anbindung an andere Entwicklungsumgebungen prinzipiell möglich.

### Was sind die Unterschiede zwischen dem CP 5613 und dem CP 5614?

Der CP 5614 verfügt zusätzlich über einen zweiten DP-Anschluss, mit dem der PC als DP-Slave an einem zweiten PROFIBUS-Netz betrieben werden kann. Zusätzlich gibt es eine Transferfunktion zur automatischen Übertragung von Slave-Daten zum übergeordneten Master.

### Unterstützt der CP 5613 und CP 5614 auch die DP-Programmierschnittstelle des CP 5412 (A2), des CP 5511 und CP 5611?

Dafür ist das Zusatzprodukt „DP-5613/Windows NT“ vorgesehen.

### Werden mehrere CPUs, Anwenderprogramme und CPs parallel unterstützt?

Mehrere CPUs in einem PC: ja.

Beachten Sie ansonsten bitte die Versionstabelle im Kapitel 14.2 der Installationsanleitung.

## Wird ein User-Watchdog unterstützt?

Beachten Sie dazu bitte die Versionstabelle im Kapitel 14.2 der Installationsanleitung.

## Sind der CP 5613 und CP 5614 OPC-fähig?

OPC-Produkte zur Unterstützung des CP 5613/CP 5614 sind in Planung.

## Welche Reaktionszeiten und Datentransferraten erreicht der CP 5613/CP 5614?

Beachten Sie dazu bitte die Mengengerüstangaben im Kapitel 14 der Installationsanleitung.

## 5.2 FAQ zur Strukturierung Ihres Anwenderprogramms

### Können DP und DPC1 parallel betrieben werden?

Ja. Der DP-Master muss aber schon im Zustand OPERATE sein, bevor DPC1-Funktionen benutzt werden können.

### Wieviele DPC1-Aufträge können parallel abgesetzt werden?

Pro Slave-Adresse können zu einem Zeitpunkt bis zu zwei DPC1-Aufträge abgesetzt werden, davon einer für lesen oder schreiben und einer für Alarm-Acknowledge.

### Welche Strukturierungsmöglichkeiten für Anwenderprogramme gibt es?

Bei den Wartemechanismen gibt es im Prinzip drei Möglichkeiten, die Sie auch miteinander kombinieren können:

- Grundsätzlich können Sie Ihr Anwenderprogramm pollen lassen, d. h. es fragt immer wieder zyklisch das Prozessabbild ab.
- Sie können Hardware-Events mit Semaphore als Wartemechanismen einsetzen.
- Sie können die Funktion DP_get_event mit einem Timeout-Wert aufrufen, um so auf Software-Events zu warten.

Außerdem können Sie Ihr Anwenderprogramm in mehrere Threads zerlegen (siehe dazu das Programmbeispiel „ExamComp“).

### Was sind die Vorteile eines nur pollenden Anwenderprogramms?

Die Programmstruktur ist relativ einfach (vorausgesetzt Ihr Programm benutzt keine anderen Schnittstellen, die Event-Handling erfordern würden, wie z. B. DPC1). Polling ist besonders auch dann gut geeignet, wenn aufgrund einer hohen Datenübertragungsgeschwindigkeit und vieler sich rasch ändernder Slaves mit einem hohen Datenaufkommen zu rechnen ist.

## Was sind die Nachteile eines nur pollenden Anwenderprogramms?

Wenn Ihr Anwenderprogramm sich beim Pollen nicht bremst, wird es die gesamte verfügbare CPU-Leistung des PC dafür verwenden, viel zu oft das Prozessabbild abzufragen. Wenn es sich zu sehr bremst, veralten die Daten. Wir empfehlen Ihnen daher, das Pollen durch eine Zeitsteuerung zu bremsen, die zu Ihrer Anlage passt.

## Was sind die Vorteile bei Hardware-Events ?

Ihr Anwenderprogramm kann schnell auf Änderungen reagieren. Es verbraucht auch, da wartend, nicht unnötig die CPU-Leistung des PC.

## Was sind die Nachteile bei Hardware-Events?

Wenn Ihr Anwenderprogramm viele sehr aktive Slaves mit Hilfe von Hardware-Events überwacht, wird der Verwaltungsaufwand zum Abholen von Events hoch, weil ständig Semaphore weitergeschaltet und durchlaufen werden. Dadurch steigt die Belastung der PC-CPU. Wir empfehlen Ihnen daher, bei sehr vielen oder sehr aktiven Slaves Hardware-Events nur für einige Slaves einzusetzen und den Rest periodisch zu pollen.

## Was sind die Vorteile einer Programmierung mit mehreren Threads?

Mit mehreren Threads erreichen Sie eine saubere Strukturierung Ihres Anwenderprogramms. Das ist besonders dann sinnvoll, wenn Sie verschiedene Vorgänge parallel fahren möchten, z. B. periodisches Pollen kombiniert mit Hardware-Events auf der einen Seite und asynchrone DPC1-Aufträge auf der anderen Seite. Diese verschiedenen Aufgaben mit ihren unterschiedlichen Dynamiken können Sie übersichtlich in zwei Threads unterbringen.

## Was sind die Nachteile einer Programmierung mit mehreren Threads?

Die Threads eines Anwenderprogramms dürfen sich z. B. beim Zugriff auf globale Variablen nicht in die Quere kommen. Datenüberschreiben und Deadlocks müssen vermieden werden. Insbesondere dürfen nicht mehrere Threads gleichzeitig auf das Prozessabbild des CP 5613 oder CP 5614 zugreifen, weil sie sonst einander die Kontrollregister zum konsistenten Lesen und Schreiben von Slave-Daten überschreiben. Dadurch könnte z. B. einem gerade lesenden Thread die Konsistenz seiner Slave-Daten entzogen werden.

## Wie kann ich mit mehreren Threads auf das Prozessabbild zugreifen?

Wir raten davon ab. Die Kontrollregister zum konsistenten Lesen und Schreiben von Slave-Daten können dann überschrieben werden, wodurch z. B. einem gerade lesenden Thread die Konsistenz seiner Slave-Daten entzogen werden kann.
Um dies zu umgehen, müssen Sie unbedingt die Threads mit einem Verriegelungsmechanismus versehen, so dass immer nur einer zu einer Zeit auf das Prozessabbild zugreifen kann. Das können Sie durch Semaphore, Mutexe etc. von Windows NT erreichen.

## Wie kann ich mit mehreren Programmen auf das Prozessabbild zugreifen?

Das geht, wenn alle beteiligten Programme den Zeiger auf das Prozessabbild mit dem Aufruf DP_release_pointer zwischendurch wieder freigeben. Bei Echtzeitbetrieb raten wir davon ab, weil die beiden Aufrufe zeitintensiv sind und Sie zusätzlich Koordinationsmechanismen zwischen den Programmen einführen müssten.

## 5.3 FAQ Checkliste für Programmierer

### Was ist bei der Programmstruktur besonders zu beachten?

Überprüfen Sie Ihre Programmstruktur auf folgende Punkte:

- Beginnen Sie Ihr Anwenderprogramm mit einem DP_start_cp, DP_open und DP_get_pointer und beenden Sie diese in allen Fällen mit DP_release_pointer, DP_close und DP_reset_cp.
- Beachten Sie, dass nach einem DP_release_pointer oder DP_close-Aufruf nicht mehr auf das Dualport-RAM zugegriffen werden darf (Pointer ungültig).
- Werten Sie alle Fehleranzeigen aus, und prüfen Sie alle Diagnosedaten.
- Prüfen Sie, ob die angegebenen Vorbedingungen zur Gültigkeit von Daten gegeben sind. Beispielsweise muss beim Lesezugriff auf Slave-Daten der Master im Zustand OPERATE oder CLEAR und der Slave READY sein. READY bedeutet, dass der Slave vom DP-Master parametriert und konfiguriert worden ist und am DP-Zyklus teilnimmt. Zusätzlich müssen aber immer die Diagnosedaten, die der Slave an den Master sendet, ausgewertet werden.
- Lesen Sie bei alarmfähigen DPC1-Slaves Diagnosedaten **nur** über die Funktion DP_fetch_alarm aus. Quittieren Sie bei Alarmen diese durch DP_alarm_ack.
- Bei nicht alarmfähigen Slaves können Diagnosedaten ebenfalls durch DP_fetch_alarm ausgelesen werden (Alternative: konsistentes Lesen der Diagnosedaten im DPR). Beide Verfahren (DP_fetch_alarm und konsistentes Lesen der Diagnosedaten im DPR) dürfen **nicht** kombiniert werden, weil sonst die gleichen Daten mehrfach angezeigt werden können.
- Werden asynchrone Aufrufe an einen Slave mit Kommunikationsfehler (siehe DP_get_result) beendet oder meldet der Aufruf DP_fetch_alarm Datenfehler, muss der entsprechende Slave durch den Aufruf DP_slv_state neu initialisiert werden (nähere Informationen sind bei den einzelnen Funktionsbeschreibungen zu finden).
- Ereignisse können in beliebiger Reihenfolge gemeldet werden; gestalten Sie Ihre Algorithmen entsprechend flexibel.
- Schicken Sie an einen Slave nicht mehrere asynchrone DPC1-Aufträge (DP_ds_read/DP_ds_write, DP_alarm_ack) parallel.
- Ist das Ergebnis eines DP_enable_event-Aufrufs abgeholt, müssen Sie ihn erneut absetzen, damit neue Ereignisse gemeldet werden können.

## Soll ich den Zeiger währen der Laufzeit meines Programms behalten?

Solange Ihr Anwenderprogramm den Zeiger auf das Prozessabbild hat (Aufruf DP_get_pointer), kann kein anderes Anwenderprogramm darauf zugreifen. In diesem Fall behalten Sie den Zeiger bis zum Programmende. Wenn Sie aber nur „sporadisch“ auf das Prozessabbild zugreifen, z. B. als Diagnoseprogramm oder wenn Sie keine Echtzeitanforderungen haben, dann sollten Sie den Zeiger nach jedem Zugriff bzw. Zyklus wieder freigeben.

## Was ist beim Zugriff auf Datenbereiche besonders zu beachten?

Das Prozessabbild ist vom Anwenderprogramm und den Slaves entkoppelt. Beachten Sie hierzu die Hinweise in Kapitel 2.7.

Überprüfen Sie Ihre Datenbereiche auf folgende Punkte:

- Dimensionieren Sie die Datenpuffer für die größtmögliche Datenlänge.
- Verketten Sie bereits im Anlauf Ihres Programms Auftragsblöcke mit gültigen Datenpuffern.
- Benutzen Sie mehrere Threads, so sollte jeder Thread seine eigenen Auftragsblöcke und Datenpuffer haben.
- Verriegeln Sie Threads gegeneinander, wenn diese gleichzeitig auf das Dualport RAM zugreifen können, um Zugriffskonflikte zu vermeiden.

## Worauf ist beim Einsatz von Hardware-Events und Semaphore besonders zu achten?

- Solange Ihr Anwenderprogramm ein Semaphor für Hardware-Events nicht durchlaufen hat, wird es nicht nochmals aufgezogen. Sie sollten also nach dem Durchlaufen durch ein Semaphor prüfen, ob mehrere Ereignisse aufgetreten sind. (Das ist bei Software-Events nicht nötig.)
- Nach dem Eintreffen eines Hardware-Events wird die auslösende Steuerbedingung zurückgesetzt, so dass Ihr Anwenderprogramm ihn erneut setzen muss.
- Ihr Anwenderprogramm muss Semaphore mit DP_delete_sema_object abmelden, nicht mit Windows-API-Funktionen.

## Wie kann ich in einem Thread auf mehrere Ereignisse gleichzeitig warten?

Erzeugen Sie für jede Ereignisart ein Semaphor (DP_create_sema_object), stoßen Sie gegebenenfalls Ihre Aufträge an und warten Sie dann an den Semaphoren durch einen Aufruf der Win32-API-Funktion „WaitForMultipleObjects“ oder „MsgWaitForMultipleObjects“.

## Wo finde ich eine Übersicht aller Fehlermeldungen?

In den Header-Dateien „5613_ret.h“ und „5614_ret.h“. Zum Aufschlüsseln verwenden Sie bitte die Funktion DP_get_err_txt.

## Worauf muss ein Anwenderprogrammierer noch achten?

- Verwenden Sie für parallele Aufträge eindeutige Order-IDs.
- Beachten Sie, dass ein ankommender Hardware-Event seine Aktivierungsbedingung löscht. Sie müssen ihn also erneut aktivieren.
- Nutzen Sie Hardware-Events nicht gleichzeitig mit periodischen Pollen für den selben Slave.

## 5.4 FAQ für Test und Inbetriebnahme Ihres Programms

### Was sind typische Fehlerquellen bei der Inbetriebnahme?

- Busstörung aufgrund von Wackelkontakten/falscher Verkabelung/fehlendem Buswiderstand sind eine recht häufige Ursache schwer reproduzierbarer Störungen.
- Falsch projektierte Slave-Typen
- Verwechselte Datenbasen
- Busparameter fehlerhaft
- Bei den Slaves falsch eingestellte Stationsadressen

### Gibt es Werkzeuge zur Fehlereinkreisung?

Benutzen Sie das in der Installationsanleitung beschriebene Werkzeug „PG/PC-Schnittstelle einstellen“ (erreichbar über das Startmenü von Windows NT). Dieses Werkzeug eignet sich hervorragend zur Fehlerdiagnose. Von diesem „PG/PC-Schnittstelle einstellen“ aus ist außerdem im Diagnoseteil unter „Erweiterte Diagnose“ das Werkzeug „DP-Trace“ erreichbar. Mit „DP-Trace“ können Sie ein Protokoll der Aufrufe ausgewählter DP-Funktionen erzeugen. Die Bedienung von DP-Trace ist in der Installationsanleitung ebenfalls beschrieben.

## 5.5 FAQ sonstige Programmierfragen

### Lassen sich auch Gruppen von Slaves für Hardware-Events nutzen?

Das können Sie mit Software-Mitteln selbst machen, z. B. wie folgt: Bei jedem Zyklusanfang aktivieren Sie die Hardware-Events aller Slaves in Ihrer Gruppe, und sobald ein Event passiert, deaktivieren Sie alle wieder bis zum Zyklusende. Diese Möglichkeit erfordert den Äquidistanzmodus der Baugruppe.

### Welche Zugangspunkte der Applikation werden bei der Installation angelegt?

Es werden folgende Zugangspunkte angelegt:

- CP_L2_1:
- CP_L2_2:

Damit der bei CP-5613 Version 1.x angebotene Name „CP5613_5614_1“ weiter verwendet werden kann, wird der folgende Zugangspunkt zusätzlich eingerichtet:

- CP5613_5614_1

### Welche Schnittstellenparametrierungen werden bei der Installation angelegt?

Für den ersten CP werden die folgenden Schnittstellenparametrierungen eingerichtet:

- CP5613_5614(MPI)
- CP5613_5614(PROFIBUS)

Für den zweiten bis vierten CP werden die folgenden Schnittstellenparametrierungen eingerichtet:

- CP5613_5614(MPI)<Board *n*>
- CP5613_5614(PROFIBUS)<Board *n*>

Für das *n* ist jeweils die CP-Nummer als Ziffer einzusetzen.

### Wie unterscheidet mein Anwenderprogramm einen CP 5614 von einem CP 5613?

Beim CP 5613 liefert der Aufruf „DPS_open“ die Anzeige zurück, dass kein Slave vorhanden ist; siehe Kapitel 4.2.2.

## Was ist bei Multiapplikationsbetrieb mehrerer DP-Master-Anwenderprogramme, die auf den gleichen CP 5613 oder CP 5614 zugreifen, besonders zu beachten?

Beim Multiapplikationsbetrieb ist eine Koordinierung der Anwenderprogramme erforderlich. Sie umfasst unter anderem folgende wichtige Punkte:

- Solange ein Anwenderprogramm einen Zeiger auf das Prozessabbild besitzt (Aufruf DP_get_pointer), kann kein anderes Anwenderprogramm darauf zugreifen. Deshalb muss ein Anwenderprogramm nach jedem Zugriff oder Zyklus diesen Zeiger wieder freigeben (Aufruf DP_release_pointer).

- Das Ändern des DP-Master-Zustands (Hochfahren und Herunterfahren des DP-Masters mittels Aufruf DP_set_mode) sollte nur von einer Applikation durchgeführt werden, um Zugriffskonflikte zu vermeiden.

- Bevor der DP-Masters vom Zustand CLEAR in den Zustand OPERATE gebracht wird, muss sichergestellt sein, dass die Ausgabedaten der DP-Slaves im Dualport RAM mit geeigneten Werten vorbelegt worden sind. Sobald der DP-Master im Zustand OPERATE ist, werden diese Daten an die (betriebsbereiten) DP-Slaves gesendet.

- Es sollte immer nur eine Applikation auf die Ausgangsdaten des gleichen Slaves schreiben. Dadurch wird vermieden, dass diese Daten von einer anderen Applikation überschrieben werden, bevor sie an den Slave gesendet worden sind.

- Semaphore gleichen Typs können, mit Ausnahme des Semaphors DP_OBJECT_TYPE_ASYNC, nur von einer Applikation verwendet werden, da sie nur einmal pro CP 5613/CP 5614 erzeugt werden.

- Events und Filter dürfen nicht von mehreren Applikationen für die gleichen Slaves verwendet werden (Stichwort: Zugriffskonflikte).

- Aktiviert ein DP-Anwenderprogramm die AUTOCLEAR-Eigenschaft (Aufruf DP_slv_state) oder startet sie den Watchdog (Aufruf: DP_watchdog), so wechselt der DP-Master selbsttätig vom Zustand OPERATE in den Zustand CLEAR, wenn eines dieser Ereignisse eintrifft. Daher müssen auch die übrigen Applikationen ständig den Master-Zustand überprüfen.

## Wie kann ich den CP 5613 oder CP 5614 mit einem Echtzeit-Kernel unter Windows NT benutzen, ohne gleich den ganzen Treiber usw. portieren zu müssen?

Realisieren Sie den CP-Anlauf und das Abmelden unter Windows NT. Dazwischen betreiben Sie den CP im Echtzeit-Kernel nur über den Zugriff auf das Prozessabbild. Der verwendete Interrupt u. a. Daten werden bei jedem Hochlauf des Windows NT-Treibers in die Struktur „info_watch.pci“ im Prozessabbild eingetragen (siehe Header-Datei „dp_5613.h“). Ein Windows NT-Anwenderprogramm kann sie von dort lesen und an einen Echtzeit-Kernel weitergeben.

## Wen kann ich bei Problemen ansprechen?

Bitte lesen Sie dazu das Kapitel „Wo Sie Hilfe bekommen“ in der Installationsanleitung zum Produkt.

Für Zusatzinformationen an Programmierer siehe auch:
http://www.ad.siemens.de/net/html_00/index.shtml

# Wo Sie Hilfe bekommen

# 6

Das folgende Kapitel nennt Ansprechpartner von SIMATIC NET:

Ansprechpartner für technische Fragen

Ansprechpartner für Schulung von Produkten von SIMATIC NET

## 6.1 Hilfe bei technischen Fragen

### Dokumentation

Themen zur Nutzung der vorliegenden Software finden Sie in den folgenden Informationsquellen:

- in der zugehörigen Papierdokumentation
- in der in die Software integrierten Hilfe (Taste F1)
- auf Textdateien der Lieferdiskette/Lieferdisketten
- in Text- und PDF-Dateien der SIMATIC NET-CD

### Ansprechpartner

Sollten Sie in den angegebenen Informationsquellen keine Antworten auf technischen Fragen zur Nutzung der beschriebenen Software erhalten, wenden Sie sich bitte an Ihren Siemens-Ansprechpartner in den für Sie zuständigen Vertretungen oder Geschäftsstellen.

Die Adressen finden Sie:

- in unserem Katalog IK 10
- im Internet (http://www.ad.siemens.de)
- in der Datei „liesmich.txt" im Hauptverzeichnis der SIMATIC NET-CD

### Häufige Fragen

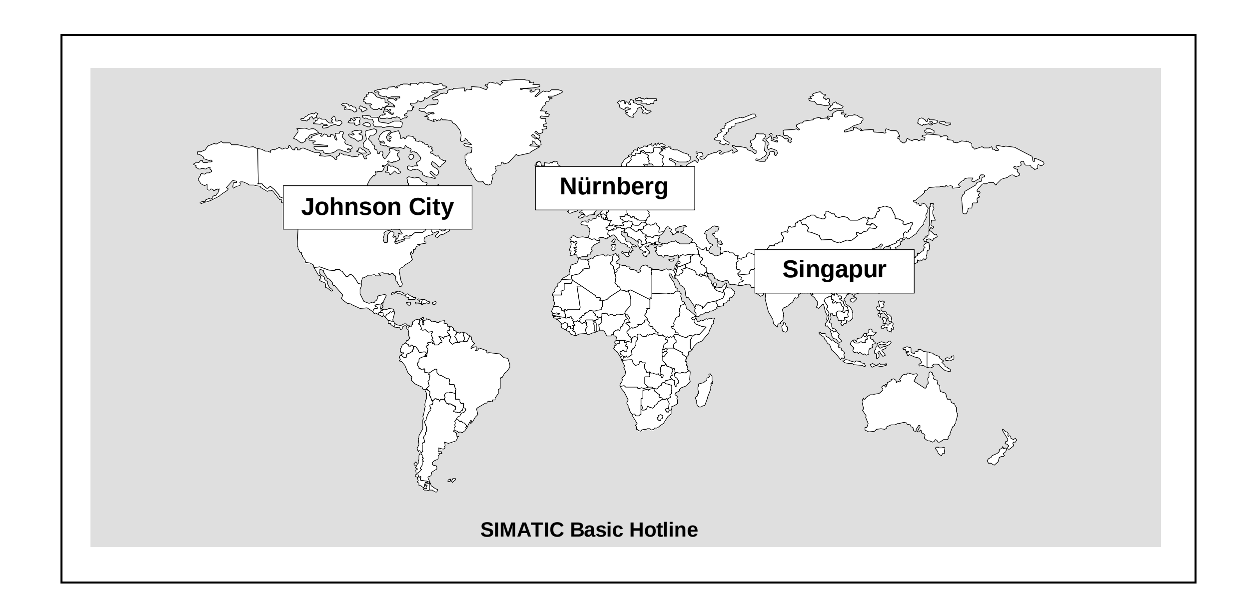
Nützliche Informationen und Antworten auf häufig gestellte Fragen bietet Ihnen unser Customer Support im Internet. Hier finden Sie im Bereich FAQ (**F**requently **A**sked **Q**uestions) Informationen rund um unser Produktspektrum.

Die Adresse der A&D-Homepage im World Wide Web des Internets lautet:

- http://www.ad.siemens.de/net

## SIMATIC Customer Support Hotline

Weltweit erreichbar zu jeder Tageszeit:




| **Nürnberg**<br>**SIMATIC BASIC-Hotline** | **SIMATIC Premium-Hotline**<br>(kostenpflichtig, nur mit SIMATIC Card) |
|---|---|
| Ortszeit: Mo bis Fr 7:00 bis 17:00 Uhr (CET)<br>Telefon: +49 (911) -895-7000<br>Fax: +49 (911) -895-7002<br>E-Mail: simatic.support@nbgm.siemens.de | Ortszeit: Mo bis Fr 0:00 bis 24:00 Uhr (CET)<br>Telefon: +49 (911) -895-7777<br>Fax: +49 (911) -895-7001 |

| **Johnson City**<br>**SIMATIC BASIC-Hotline** | **Singapur**<br>**SIMATIC BASIC-Hotline** |
|---|---|
| Ortszeit: Mo bis Fr 8:00 bis 17:00 Uhr<br>Telefon: +1 423 461-2522<br>Fax: +1 423 461-2231<br>E-Mail: simatic.hotline@sea.siemens.com | Ortszeit: Mo bis Fr 8:30 bis 17:30 Uhr<br>Telefon: +65 740-7000<br>Fax: +65 740-7001<br>E-Mail: simatic.hotline@sae.siemens.com.sg |

## SIMATIC Customer Support Online-Dienste

Das SIMATIC Customer Support bietet Ihnen über die Online-Dienste umfangreiche, zusätzliche Informationen zu den SIMATIC-Produkten:

- Allgemeine, aktuelle Information erhalten Sie:
  - im Internet (http://www.ad.siemens.de/simatic)
  - über Fax-Polling
- Aktuelle Produktinformation und Downloads, die beim Einsatz nützlich sein können erhalten Sie:
  - im Internet (http://www.ad.siemens.de/support/html-00).
  - über das Bulletin Board System (BBS) der SIMATIC Customer Support Mailbox in Nürnberg (Tel.: +49 (911) - 895-7100).

    Verwenden Sie zur Anwahl der Mailbox ein Modem mit bis zu 28,8 kbit/s, V.34 (Parameter: 8, N, 1, ANSI) oder ISDN (Parameter: x.75, 64 kbit/s).

## 6.2 Ansprechpartner für Schulung von SIMATIC NET

### SIMATIC Trainings-Center

Wenden Sie sich bitte an Ihr regionales Trainings-Center oder an das zentrale Trainings-Center in D-90327 Nürnberg, Tel. 0911-895-3154.

# Glossar

# 7

## Anlage

Gesamtheit aller elektrischen Betriebsmittel - Zu einer Anlage gehören u. a. Speicherprogrammierbare Steuerung, Geräte für Bedienen und Beobachten, Bussysteme, Feldgeräte, Antriebe, Versorgungsleitungen.

## Ansprechüberwachungszeit

Eine im DP-Slave einstellbare Überwachungszeit zur Ausfallerkennung des zugeordneten DP-Masters.

## Ausgabedaten

Hier: Die Daten, die der DP-Master zyklisch an die Slaves schickt.

## AUTOCLEAR

1. Projektierungseigenschaft eines DP-Slave - Der Master geht nach Ausfall dieses Slave in den AUTOCLEAR-Zustand.

2. Mit DP_CLEAR bedeutungsgleicher Zustand des DP-Masters, wenn er aufgrund der AUTOCLEAR-Eigenschaft eines Slave erreicht wird.

## Bitzeiten

Zeit, die ein Bit zur Übertragung am Bus braucht, also der Kehrwert der Datenübertragungsgeschwindigkeit.

## Busparameter

Busparameter steuern das Übertragungsverhalten am Bus - Jeder Teilnehmer an PROFIBUS muss mit den Busparametern anderer Teilnehmer übereinstimmende Busparameter verwenden.

### c_ref

Kennung zur **I**dentifikation von Verbindungen zu Slaves bei DPC1-Aufrufen.

### COM PROFIBUS

Projektierungswerkzeug zur Beschreibung der Kommunikationsteilnehmer und der Busparameter.

### CP

**C**ommunication **P**rocessor - Kommunikationsbaugruppe/Netzwerkkarte für den Einbau in Rechner oder Automatisierungssysteme.

### CPU

**C**entral **P**rocessor **U**nit - hier: Prozessor des PC

### CPU-Last

Belastung der CPU des PC - hier: durch DP-Kommunikation

### Data_Ex

Slave ist bereit für den Produktivbetrieb

### Datenübertragungsgeschwindigkeit

Übertragungsrate am Bus (Einheit: bit/s). Ein Busparameter für PROFIBUS. Die Angabe bzw. Wahl der Datenübertragungsgeschwindigkeit hängt von verschiedenen Randbedingungen ab, wie beispielsweise Entfernung.

### DB

**D**aten**b**asis - Die lokale Datenbasis beschreibt das Kommunikationsnetz aus Sicht des eigenen Systems.

### Deadlock

Wenn bei mehreren parallelen Prozessen (hier: Threads) A auf B wartet und B auf A.

### Dezentrale Peripherie

Ein- und Ausgabebaugruppen, die dezentral von der CPU (Zentraleinheit der Steuerung) eingesetzt werden. Die Verbindung zwischen dem Automatisierungsgerät und der Dezentralen Peripherie erfolgt über das Bussystem PROFIBUS. Automatisierungsgeräten wird der Unterschied zu lokalen Prozessein- oder Prozessausgaben verdeckt.

### Dienste

Angebotene Leistungen eines Kommunikationsprotokolls.

### DP

Dezentrale Peripherie, Kommunikationsprotokoll für PROFIBUS gemäß EN 50 170 Volume 2.

### DP-Base

Name für die DP-Programmierschnittstelle des CP 5613/CP 5614, im Gegensatz zur DP-Lib-Schnittstelle des CP 5412 (A2), CP 5611 und CP 5511.

### DPC1

Erweiterung von DP um azyklische Read- und Write-Aufträge und Alarme zwischen zyklischem DP-Master und Slave.

### DPC2

Erweiterung von DP um Verbindungssteuerung sowie Read- und Write-Aufträge von einem nicht-zyklischen Master.

### DP-V1

DP-Erweiterungen - Oberbegriff für DPC1 und DPC2

### DP-Master

Ein Teilnehmer mit Master-Klasse-1-Funktion bei PROFIBUS DP - Der DP-Master wickelt den Nutzdatenverkehr mit den ihm zugeordneten DP-Slaves ab.

## DP-Master Klasse 1

siehe DP-Master

## DP-Master Klasse 2

Optionaler Diagnose-Master – Der Diagnose-Master dient zur Überwachung des DP-Master Klasse 1 und der DP-Slaves.

## DP-Slave

Ein Teilnehmer mit Slave-Funktion bei PROFIBUS DP.

## DP-Subnetz

PROFIBUS-(Sub)Netz, an dem nur Dezentrale Peripherie betrieben wird.

## DP-Subsystem

Ein DP-Master und alle DP-Slaves, mit denen dieser DP-Master Daten austauscht.

## Dualport RAM

**D**ual **P**ort **R**andom **A**ccess **M**emory - Gestattet den gleichzeitigen Zugriff von zwei Rechnereinheiten (CP und CPU) auf einen Speicherbaustein (RAM).

## E/A-Modul

DP-Slaves können modular aufgebaut sein. Ein DP-Slave besitzt mindestens ein DP-E/A-Modul.

## E/A-Typ

DP-E/A-Typ bezeichnet ein DP-E/A-Modul. Zu unterscheiden sind:

- Eingabemodul
- Ausgabemodul
- Ein-/Ausgabemodul

## Eingabedaten

Hier: Die Daten, die der DP-Master zyklisch von den Slaves liest.

## Event

Hier: Ein Ereignis, das der CP 5613/CP 5614 dem Anwenderprogramm melden kann. Es gibt Hardware-Events und Software-Events.

## Fast Logic

Eigenschaft des CP 5613/CP 5614: ein Eingabewert eines Slave kann überwacht werden. Wenn es einen bestimmten Wert annimmt, wird ein Ausgabedatum eines anderen Slave gesetzt.

## FDL

**F**ieldbus **D**ata **L**ink - Schicht 2 bei PROFIBUS

## FREEZE-Modus

Der FREEZE-Modus ist eine DP-Betriebsart, bei der von allen (oder von einer Gruppe von) DP-Slaves zeitgleich Prozessdaten erfasst werden. Der Erfassungszeitpunkt wird durch das FREEZE-Kommando (das ist ein Steuertelegramm zur Synchronisation) signalisiert.

## Gap-Aktualisierungsfaktor

Ein freier Adressbereich zwischen zwei aktiven Teilnehmern wird zyklisch von dem Teilnehmer mit der kleineren PROFIBUS-Adresse durchsucht um festzustellen, ob ein weiterer Teilnehmer in den logischen Ring aufgenommen werden möchte. Die Zykluszeit für diese Überprüfung wird bestimmt durch:

Gap-Aktualisierungsfaktor x Target rotation time [ms]

## Gruppenidentifikation

DP-Slaves können über eine Gruppenidentifikation einer oder mehreren Gruppen zugewiesen werden. Die DP-Slaves können dann über die Gruppenidentifikation bei der Übertragung von Steuertelegrammen gezielt angesprochen werden.

## Höchste PROFIBUS-Adresse

Ein Busparameter für PROFIBUS. Gibt die höchste PROFIBUS-Adresse eines aktiven Teilnehmers an PROFIBUS an. Für passive Teilnehmer sind PROFIBUS-Adressen größer als HSA zulässig (Wertebereich: HSA 1 bis 126).

## HSA

**H**ighest **S**tation **A**ddress - einer der Busparameter. Gibt die höchste im Netz verwendete Teilnehmeradresse an.

## Hardware-Event

Event, dessen Eintreffen von der Hardware des CP überwacht wird (bei Zyklusbeginn, Zyklusende, Datenänderung, Eintreffen von Diagnose- und Eintreffen einer Fast-Logic-Bedingung).

## Indication

Nachricht von einem remoten Teilnehmer.

## Intel-Format

Zahlen werden im Intel-Format gespeichert, wenn niederwertige Bytes zuerst (d. h. auf niedrigen Adressen) abgelegt werden.

## ISO

**I**nternational **S**tandard **O**rganization **-** **I**nternationale Organisation mit Sitz in Genf zur Schaffung allgemeiner Normen, vor allem auf dem Gebiet des Datenübertragungsbereichs.

## LAN

**L**ocal **A**rea **N**etwork - Lokales Netzwerk zur direkten Verbindung von Computern.

## LSB

**L**east **S**ignificant **B**it - niederwertigstes Bit eines Bytes.

## Master

Aktiver Teilnehmer an PROFIBUS, der unaufgefordert Telegramme senden kann, wenn er im Besitz des Token ist.

## Maximum Station Delay

Ein Busparameter für PROFIBUS - Die Maximum Station Delay (max. $T_{SDR}$) gibt die größte, bei einem der Teilnehmer im Subnetz benötigte Zeitspanne an, die zwischen dem Empfang des letzten Bits eines unquittierten Telegramms bis zum Senden des ersten Bits des nächsten Telegramms vergehen muss. Ein Sender darf nach dem Senden eines unquittierten Telegramms erst nach Ablauf der Zeitspanne max. $T_{SDR}$ ein weiteres Telegramm senden.

## Minimum Station Delay

Ein Busparameter für PROFIBUS - Das „Minimum Station Delay“ (min. $T_{SDR}$) gibt die Zeitspanne an, die der Empfänger eines Telegramms bis zum Senden der Quittung oder bis zum Senden eines weiteren Telegramms mindestens warten muss. Die min. $T_{SDR}$ richtet sich nach der größten, bei einem Teilnehmer im Subsystem benötigten Zeitspanne zur Entgegennahme einer Quittung nach dem Senden des Telegramms.

## min $T_{SDR}$

Siehe „Minimum Station Delay“

## Motorola-Format

Zahlen werden im Motorola-Format gespeichert, wenn höherwertige Bytes zuerst (d. h. auf niedrigen Adressen) abgelegt werden.

## MSAC_C1

„Master Slave Acyclic Class 1“ - Name für DPC1 in der DP-V1-Normbeschreibung.

## MSAC_C2

„Master Slave Acyclic Class 2“ - Name für DPC2 in der DP-V1-Normbeschreibung.

## MSB

**M**ost **S**ignificant **B**it - höchstwertigstes Bit eines Bytes.

## MSCY_C1

„Master Slave cyclic class 1“ - Name für den normalen DP-Master-Betrieb in der DP-V1-Normbeschreibung.

## Netz

Ein Netz besteht aus einem oder mehreren verknüpften Subnetzen mit einer beliebigen Zahl von Teilnehmern. Es können mehrere Netze nebeneinander bestehen. Für jedes Subnetz gibt es eine gemeinsame Knotentabelle.

## PC

**P**ersonal **C**omputer

## PG

**P**rogrammier**g**erät - Programmiergerät für die Produktfamilie SIMATIC der Siemens AG; wird eingesetzt zum Programmieren, zur Projektierung, bei der Wartung und im Service.

## PNO

PROFIBUS-Nutzer-Organisation

## PROFIBUS

Feldbus nach EN 50 170 Vol. 2 (DIN 19245).

## PROFIBUS-Adresse

Die PROFIBUS-Adresse ist eine eindeutige Kennung eines an PROFIBUS angeschlossenen Teilnehmers. Zur Adressierung eines Teilnehmers wird die PROFIBUS-Adresse im Telegramm übertragen.

## PROFIBUS DP

PROFIBUS DP EN 50 170 Vol. 2 (DIN 19245 T1 + T3) ist eine Richtlinie der PROFIBUS Nutzerorganisation (PNO) für den Datenaustausch mit dezentralen Peripheriegeräten.

## Protokoll

Verfahrensvorschrift für die Übermittlung in der Datenübertragung - Mit dieser Vorschrift werden sowohl die Formate der Nachrichten als auch der Datenfluss bei der Datenübertragung festgelegt.

## Prozessabbild

Hier: verwendet für ein Dualport RAM des CP 5613/CP 5614, auf den Anwenderprogramme direkt zugreifen können und wo u. a. die jeweils aktuellen Daten der Slaves liegen.

## READY-Time

Ein Busparameter für PROFIBUS - Die READY-Time ist die Zeit, in der ein aktiver Teilnehmer nach der Aussendung eines Aufrufs für eine Quittung oder Antwort empfangsbereit sein muss.

## Reorganisation Token-Ring

Alle Master am PROFIBUS (PROFIBUS) bilden einen logischen Token-Ring. Innerhalb dieses Token-Rings wird die Sendeberechtigung (Token) von Station zu Station weitergegeben. Wird nun die Übertragung des Token gestört oder wird ein Master vom Token-Ring entfernt, so führt dies bei der Token-Weitergabe zu einem Fehler (Token wird von dieser Station nicht angenommen), was eine Ausgliederung dieser Station aus dem Token-Ring zur Folge hat. Die Anzahl der Ausgliederungen werden im internen Token_error_counter gezählt. Erreicht dieser Zähler einen oberen Grenzwert, dann wird der logische Token-Ring neu aufgebaut (reorganisiert).

## S7-AG

Abkürzung für ein Automatisierungsgerät der Produktfamilie SIMATIC der Siemens AG.

## SAP

**S**ervice **A**ccess **P**oint - Zugangspunkt zum PROFIBUS innerhalb einer Station

## Semaphor

Wartestellenmechanismus zur Synchronisation mehrerer Programme, z. B. in Windows NT.

## Setup Time

Ein Busparameter für PROFIBUS - Die „Setup Time“ gibt den Mindestzeitabstand zwischen dem Empfang einer Quittung bis zum Senden eines neuen Aufruftelegramms durch den Sender an.

## SIMATIC NET

Siemens Network and Communication - Produktbezeichnung für Netze und Netzkomponenten bei Siemens.

## Slot Time

Ein Busparameter für PROFIBUS - Die Slot Time (TSL) ist die Überwachungszeit eines Senders eines Telegramms auf die Quittung des Empfängers.

## Software-Event

Hier: Event, der mit der Funktion DP_get_result abgeholt werden kann.

## SYNC-Modus

Der SYNC-Modus ist eine DP-Betriebsart, bei der mehrere oder alle DP-Slaves zu einem bestimmten Zeitpunkt Daten an ihre Prozessausgänge übergeben. Der Übergabezeitpunkt wird durch das SYNC-Kommando (das ist ein Steuertelegramm zur Synchronisation) signalisiert.

## Target Rotation Time

Ein Busparameter für PROFIBUS - Der Token ist die Sendeberechtigung für einen Teilnehmer an PROFIBUS. Ein Teilnehmer vergleicht eine von ihm gemessene Token-Umlaufzeit mit der Target rotation time und steuert davon abhängig das Senden hoch- und niederpriorer Telegramme.

## Telegramm

Nachricht eines PROFIBUS-Teilnehmers an einen anderen.

## Telegramm-Header

Ein Telegramm-Header besteht aus einer Kennung des Telegramms sowie der Quell- und Zielteilnehmeradresse.

## Telegramm-Trailer

Der Telegramm-Trailer besteht aus einer Prüfsumme und der Endekennung des Telegramms.

## Thread

Parallel laufender Subprozess

## Treiber

Software, die dem Datenaustausch von Anwenderprogrammen mit dem CP dient.

## User-Watchdog

Watchdog zur Überwachung des DP-Anwenderprogramms

## Watchdog

Mechanismus zur Überwachung der Betriebsbereitschaft

# Stichwortverzeichnis

# 8

❑